FELIX GARCIA MERAYO

De IBM ESPAÑA

Doctor en Informática
Ldo. en Ciencias Matemáticas
Ldo. en Informática
Prof. de la Facultad de Informática
Universidad Politécnica de Madrid

Programación en FORTRAN 77

1991

PARANINFO S.A

TERCERA EDICION
CORREGIDA Y AMPLIADA

MADRID

Ilustración de portada
por cortesía de ERDISA

© FELIX GARCIA MERAYO

© EDITORIAL PARANINFO, S.A.
Magallanes, 25 - 28015 Madrid
Teléfono: 4463350 - Fax: 4456218

Reservados los derechos para todos los países. Ninguna parte de esta publicación, incluido el diseño de la cubierta, puede ser reproducida, almacenada o transmitida de ninguna forma, ni por ningún medio, sea éste electrónico, químico, mecánico, electro-óptico, grabación, fotocopia o cualquier otro, sin la previa autorización escrita por parte de la Editorial.

Impreso en España
Printed in Spain

ISBN: 84-283-1818-2

Depósito legal: M. 42.364.—1990

PARANINFO S.A. Magallanes, 25 - 28015 MADRID (11303/42/85)

Artes Gráficas BENZAL, S.A., Virtudes, 7 - 28010 MADRID

Indice de materias

1. ELEMENTOS DEL LENGUAJE 15

1.1. FORTRAN 77 15
1.2. Conceptos básicos 17
1.3. Caracteres, constantes y variables 20
1.4. Algunos aspectos de la memoria principal 25
1.5. Constantes enteras 27
1.6. Constantes reales 28
1.7. Constantes lógicas 32
1.8. Constantes carácter 32
1.9. Constantes doble precisión 33
1.10. Variables enteras y reales 34
1.11. Variables lógicas 34
1.12. Variables carácter 35
1.13. Variables doble precisión 36
1.14. Cuadro resumen 36
1.15. La hoja de codificación FORTRAN 38
1.16. Las sentencias FORTRAN 42

Ejercicios resueltos 44

2. SENTENCIAS BASICAS 49

2.1. Operadores aritméticos 49
2.2. Expresiones aritméticas 50
2.3. Conjunto de datos 55
2.4. Formación de subíndices 58
2.5. Declaración de conjuntos 59
2.6. Disposición de los conjuntos en memoria 60
2.7. Sentencia de asignación aritmética 65

2.8. Sentencias PROGRAM, STOP y END 68
2.9. Sentencia GO TO incondicional 71
2.10. Sentencia GO TO calculada 73
2.11. Sentencia IF aritmético 75
2.12. Sentencia DO 80
2.13. Sentencias DO en nido 84
2.14. Reglas sobre el uso del DO 88
2.15. Sentencia CONTINUE 92
2.16. Sentencia PAUSE 94
Ejercicios resueltos 96

3. LECTURA Y ESCRITURA DE LA INFORMACION. FICHEROS 108

3.1. Entradas y salidas 108
3.2. Campo, registro, fichero 110
3.3. Acceso a un fichero 114
3.4. Operaciones sobre un fichero 116
3.5. Sentencias READ básica y PRINT 117
3.6. Sentencias de E/S para acceso secuencial 126
3.7. Sentencia READ general 127
3.8. Sentencia WRITE 131
3.9. Sentencias READ y WRITE sin formato 135
3.10. Sentencias END FILE, REWIND y BACKSPACE 136
3.11. Sentencias de E/S para acceso directo 138
3.12. DO ímplicito en listas de E/S 140
3.13. La instrucción OPEN 144
3.14. La instrucción CLOSE 149
3.15. La instrucción INQUIRE 151
Ejercicios resueltos 155

4. CODIGOS 164

4.1. Sentencia y códigos de formato 164
4.2. Código I 168
4.3. Código F 173
4.4. Códigos E, D y G 177
4.5. Código L 183
4.6. Código A 184
4.7. Códigos X, T, TR y TL 187
4.8. Código de fin de registro 190
4.9. Códigos para constantes CARACTER 193
4.10. Control de carro 194

4.11. Códigos SP, SS y S 195
4.12. Códigos BN y BZ 196
4.13. Código P 198
4.14. Código (:) 199
4.15. Usos diversos del FORMAT 200
Ejercicios resueltos 204

5. ELEMENTOS LOGICOS. ESTRUCTURAS DE CONTROL 216
5.1. Operadores y expresiones de relación 216
5.2. Operadores lógicos 218
5.3. Expresiones lógicas 221
5.4. Prioridad de operaciones al emplear expresiones lógicas 223
5.5. Sentencia de asignación lógica 225
5.6. Sentencia IF lógico 226
5.7. Bloque IF 228
Ejercicios resueltos 236

6. OTRAS SENTENCIAS FORTRAN 241
6.1. Otras sentencias de tipo: INTEGER, REAL e IMPLICIT 241
6.2. Constantes y variables complejas 244
6.3. Expresiones complejas 246
6.4. Operaciones de E/S de datos complejos 250
6.5. Expresiones tipo CARACTER 251
6.6. Sentencia de asignación CARACTER 256
6.7. Sentencia PARAMETER 259
6.8. Sentencia DATA 263
6.9. Sentencia GO TO asignado 266
Ejercicios resueltos 268

7. PROGRAMACION MODULAR: FUNCIONES 278
7.1. Unidades programa 278
7.2. Subprogramas 279
7.3. Subrutinas y funciones 280
7.4. Funciones intrínsecas 283
7.5. Funciones de sentencia 294
7.6. Funciones externas: Sentencias FUNCTION y RETURN 299
7.7. Sentencias EXTERNAL e INTRINSIC 308
7.8. Efectos borde 310
Ejercicios resueltos 311

8. PROGRAMACION MODULAR: SUBRUTINAS 317

8.1. Tipos de llamada a los procedimientos 317
8.2. Subrutinas: Sentencias SUBROUTINE y CALL 319
8.3. Utilización de conjuntos en funciones y subrutinas 323
8.4. Dimensiones ajustables 327
8.5. Entradas y salidas múltiples: Sentencia ENTRY 328
8.6. Sentencia SAVE 331
8.7. Sentencia EQUIVALENCE 332
8.8. Sentencia COMMON 337
8.9. Utilización conjunta de sentencias COMMON y EQUIVALENCE . 340
8.10. Sentencia COMMON con uso de etiquetas 341
8.11. Subprograma BLOCK DATA 345

Ejercicios resueltos 346

Apéndice 1. ORGANIGRAMAS-SIMBOLOS EMPLEADOS EN EL TEXTO 352

Apéndice 2. RESUMEN DE LAS SENTENCIAS FORTRAN 77 ... 355

Apéndice 3. FICHEROS EXTERNOS E INTERNOS 358

Apéndice 4. ESTRUCTURA DE LAS SENTENCIAS FORTRAN 77 363

Apéndice 5. ALGORITMOS PROGRAMADOS 368

Bibliografía 393

Indice alfabético 395

Prefacio

Me proporciona un gran placer el escribir la introducción de este nuevo trabajo sobre FORTRAN.

El libro que ahora aparece describe de una forma clara y elegante el último standard publicado para el lenguaje FORTRAN: F-77.

Durante muchos años, y desde su primera aparición allá por los años 55, FORTRAN se concebió, y muy notoriamente, como un lenguaje orientado únicamente a las aplicaciones numéricas. Se ha convertido luego en una herramienta capaz de manejar también caracteres y ficheros y en el momento actual se utiliza ampliamente en aplicaciones relacionadas con los gráficos. Lejos de ser un lenguaje estático y de valor decreciente, hoy día el lenguaje de programación FORTRAN muestra un vigoroso desarrollo que augura un presente con vitalidad y un continuo uso en un amplio espectro de las aplicaciones de ordenador.

La evolución de los grandes sistemas de cálculo utilizando cada vez mayor cantidad de memoria principal, permite un empleo sencillo de grandes volumenes de datos. La potencia del lenguaje FORTRAN capacita al programador para lograr una buena utilización de las posibilidades de los grandes sistemas, particularmente en todos aquellos casos relacionados con el análisis numérico. Sin embargo, sería erroneo el considerar al lenguaje FORTRAN ligado únicamente a cálculos de gran envergadura ejecutados sobre máquinas con alto grado de posibilidades. Una gran parte de las aplicaciones desarrolladas con este lenguaje para minis y microprocesadores demuestra la importancia del FORTRAN a lo largo y ancho de toda clase de cálculos.

Existen dos grupos de usuarios para los que el FORTRAN representa un especial interés: el grupo compuesto por científicos, por personas que trabajan en la Universidad o los que hacen investigación y aquel otro donde se incluirían los ingenieros de todo tipo. La transformación del lenguaje FORTRAN en un moderno lenguaje de cálculo de carácter general les ha provisto a todos ellos con la capacidad de poder realizar sus tareas de una forma más eficiente actualizando sus herramientas y técnicas de cálculo y proporcionando la suficiente fuerza, incluso en problemas más complejos en los que siempre existe un creciente volumen de datos.

Quiero recomendar este libro no sólo por su claridad y por constituir un buen trabajo escrito para programadores FORTRAN, sino también por considerarlo como un valioso libro de texto para cursos sobre lenguajes de programación. Debido a los muchos ejemplos y ejercicios que el autor ha incluido, lo hacen particularmente preciado en este contexto.

Raymond EVANS
Director
IBM, Centro de Marketing
Científico y de Ingeniería
PARIS

Preface

It gives me great pleasure to write the introduction to this new work on FORTRAN.

The book describes clearly and elegantly the latest published standard for the FORTRAN language: F-77. Over the years since its first appearance in 1955 FOTRAN has evolved markedly from its origins as a language used solely for numerically orientated applications. It has acquired the ability to handle character and file manipulation and it is also widely used for graphics applications. Far from being a static language of decreasing value in todays computing environment FORTRAN shows a vigourous development which augurs well for its present vitality and for continuing use across a broad spectrum of computing.

The evolution of large computer systems with ever growing amounts of main storage permits the easy manipulation of very large volumes of data. The power of FORTRAN allows the programmer to make good use of the facilities of large computer systems – particularly where numerical analysis work is concerned. However, it would be wrong to regard FORTRAN nearly as a language for large scale computation on machines of very high perfomance. The many implementation of the language available for mini and micro processors testify to the importance of FORTRAN across the whole range of computing.

There are two groups of users for whom FORTRAN is of special interest. They are the Scientific, Academic and Research Community and, of course, the Engineering Community. The transformation of FORTRAN into a modern general-purpose computing language has enabled them to perform their tasks more efficiently and to update their tools and techniques to cope with ever more complex problems involving ever growing volumes of data.

I recommend this book both as a clear and well written reference work for FORTRAN programmers and also as a valuable language text book for computer science courses. The many examples and exercises that are included in it make it particularly valuable in this context.

Raymond EVANS
Manager
IBM, Scientific and
Engineering Marketing Centre
PARIS

Nota del autor

Existen muchos libros que tratan del lenguaje FORTRAN. Ahora bien; es preciso recomendar al lector que tome cuidado de no adquirir una obra en la que se exponga un FORTRAN anticuado. El primer objetivo que nos hemos marcado al redactar este texto ha sido el de recoger el más actual, el cual se conoce con el nombre de FORTRAN 77 y que responde a la norma X3.9-1978, enunciada completamente en "ANSI X3.9-1978: Programming Language FORTRAN". Dentro de los dos niveles de estandarización contenidos en la citada norma, nosotros hemos recogido en este texto la más amplia y completa. De esa manera el lector podrá abstraer y quedarse con aquella materia que más se acerque a sus necesidades y requerimientos. Con ello queremos significar que las lecciones que constituyen este libro serán útiles, tanto a los usuarios de grandes ordenadores como a los de otros más pequeños e incluso a los que manejen micros o posean un ordenador personal en su casa con el que iniciarse en pequeñas aplicaciones y programas.

Otro objetivo que nos hemos propuesto, y esperamos conseguirlo, ha sido el de desarrollar la habilidad del lector de manera que le sirvamos de guía para el diseño y desarrollo de algoritmos capaces de ser ejecutados por un sistema de cálculo. Desde este punto de vista, el libro puede ser útil tanto para el universitario, como para el profesional; tanto para el ingeniero o científico, como para el estudioso de la Informática a nivel particular y doméstico.

En este volumen hemos recogido multitud de ejemplos para aclarar cada uno de los temas tratados; ejercicios resueltos con su respuesta, de manera que resulte particularmente útil a todas aquellas personas que deseen un método autodidáctico de aprendizaje.

Hemos incluido también distintos programas, distribuidos a lo largo del libro en dificultad gradual. Con ellos, el lector podrá adquirir la necesaria experiencia para escribir sus propias aplicaciones.

Será de suma importancia el trabajar con los ejercicios y programas, principalmente cuando el lenguaje FORTRAN sea el primer lenguaje científico con el que se adentre en el campo de los ordenadores.

Quiero reconocer desde aquí el ánimo que en todo momento he recibido de amigos y compañeros de profesión, como Raymond Evans, Rafael Portaencasa, Antonio Insua, Javier Sanchis, para acometer y realizar esta pequeña obra. Y a mi editor PARANINFO por haberme ofrecido la posibilidad de aparecer nuevamente en el mundo informático después de la experiencia de mi anterior libro "El lenguaje FORTRAN".

Por último quiero dedicar este trabajo a mis hijos Mª Paloma, Eduardo y Mª Victoria, alguno de los cuales también ama la Informática.

Felix García Merayo

Madrid, París, 1986

Nota del autor a la tercera edición

La edición de un libro técnico presenta, en general, muchos problemas de transcripción desde el original hasta obtener el producto acabado. Es necesario un trabajo detenido y minucioso para evitar y suprimir toda clase de erratas. Creemos que en esta tercera edición hemos conseguido este objetivo totalmente, gracias, en parte, a nuestros alumnos y compañeros que nos han ido facilitando consejos y observaciones muy prácticos.

Esta nueva edición contiene dos nuevos apéndices. Con el primero de ellos hemos querido ampliar el que actualmente ya figuraba con el título "Resumen de las Sentencias FORTRAN 77", en el sentido de dar con ellas una información detallada, no sólo en lo referido a la sintaxis, sino también a la semántica de cada sentencia FORTRAN, incluyendo el análisis de sus componentes.

En el segundo apéndice de los mencionados se ha recogido una serie de algoritmos programados y ejecutados con los que poder ejercitarse en la práctica de las sentencias estudiadas en el texto. La citada algorítmica ha sido ensayada sobre un ordenador personal y siempre haciendo uso de las sentencias más usuales y comunes.

Félix García Merayo

Madrid, Septiembre 1990

1

Elementos del lenguaje

Dios creó los números enteros.
Todo lo demás es obra del hombre.

KRONECKER *(1823-1891)*

1.1. FORTRAN 77

Desde un punto de vista elemental, los ordenadores son dispositivos muy simples, capaces de realizar un número pequeño de operaciones sencillas y siempre siguiendo un camino predeterminado, es decir, de acuerdo con unas instrucciones que el hombre se encarga de preparar. Como cada instrucción u orden a ejecutar por la máquina es muy simple y muy limitada en cuanto a sus acciones, una suma, una resta, una comparación, ese camino a seguir por la máquina, denominado programa, se compone frecuentemente de un gran número de esas instrucciones. Una vez ejecutadas por el ordenador, el conjunto de órdenes diseñadas por el hombre o programador, se obtendrá el resultado objeto del programa en cuestión.

El conjunto de instrucciones que el programador puede emplear para definir un problema capaz de ser "entendido" por un ordenador, recibe el nombre de *lenguaje de programación*.

A principio de los años 50, los lenguajes de programación eran lenguajes muy próximos a la máquina, es decir, lenguajes constituidos por instrucciones que el ordenador "entendía" y ejecutaba fácil y casi directamente.

Este tipo de lenguaje representaba, sin embargo, serias dificultades para el programador, que debía aprender y recordar una multitud de reglas y palabras mnemotécnicas para diseñar operaciones tan sencillas como la multiplicación o la división. Estos lenguajes se conocen con el nombre de *código de máquina* y otras veces *lenguajes ensambladores*.

Los lenguajes de programación evolucionaron y con el paso del tiempo los programadores han llegado a disponer de otro tipo de len-

guaje que podría denominarse próximo al hombre o próximo a la aplicación que va a diseñarse. Estos lenguajes se conocen actualmente con el nombre de *lenguajes de alto nivel* y ello significa que son fácilmente utilizables por el hombre, porque están muy próximos a su propio lenguaje o al lenguaje empleado en una determinada ciencia, como las Matemáticas, por ejemplo.

Este es el caso del lenguaje de programación FORTRAN, denominado así por derivarse de las palabras inglesas FORmula TRANslation. El significado de ese vocablo es el siguiente: el programador escribirá su programa en un lenguaje de alto nivel, semejante al empleado en fórmulas o expresiones matemáticas o algebráicas y ese programa será luego traducido a código de máquina, automáticamente, por el propio ordenador.

Resumiendo: el programador diseña su aplicación, empleando el conjunto de instrucciones que pone a su disposición el lenguaje FORTRAN; el ordenador traduce ese conjunto a su propio lenguaje (lenguaje de máquina) y luego ejecuta esa traducción para obtener los resultados que se buscaban.

Desde su aparición, el lenguaje FORTRAN se ha extendido rápidamente. Los programadores encontraron en él una herramienta útil y sencilla para escribir primero aplicaciones matemáticas, científicas, técnicas y luego incluso financieras y comerciales.

Podría decirse que cualquier persona con una inteligencia media y con interés o necesidad de diseñar aplicaciones de cualquier tipo, puede aprender y emplear este lenguaje de alto nivel y todo ello, utilizando un ordenador de gran capacidad, un microordenador o un ordenador personal, pero en todo caso sin necesidad de poseer un conocimiento profundo del propio ordenador.

El primer lenguaje FORTRAN estuvo disponible hacia 1955 y fue diseñado y desarrollado en Estados Unidos por la compañía IBM. Fueron las principales personas envueltas en este proyecto: R. Nelson, I. Ziller, L. Haibt, H. Herrick, P. Sheridan, D. Sayre, J. Backus, R. Goldberg y G. Mitchell.

El lenguaje FORTRAN fue rápidamente adoptado por otros muchos fabricantes de ordenadores y actualmente se considera como un lenguaje de ordenador orientado a resolver un amplio y general espectro de aplicaciones.

En el año 1966 el lenguaje FORTRAN fue sometido a su primera estandarización por American National Standards Institute –ANSI– con lo que se fijaban las reglas del lenguaje, lo cual facilitaba el que los programas que se escribieran podrían ser independientes de las máqui-

nas, donde más tarde se procesaran o ejecutaran, es decir, podrían utilizarse en cualquiera de ellas.

Esta versión producida de acuerdo con ANSI, se denominó FORTRAN IV o FORTRAN 66.

Más tarde, en el año 1977, se completó una versión nueva del FORTRAN mucho más evolucionada que la anterior, apareciendo una nueva estandarización que ha pasado a denominarse FORTRAN 77. El alcance completo de esta nueva versión puede encontrarse en el documento "ANSI X3.9-1978: Programming Language FORTRAN", publicado por el American National Standards Institute en Nueva York, U.S.A.

Esta publicación sobre la estandarización describe dos niveles o categorías del lenguaje FORTRAN: uno de ellos se denomina simplemente FORTRAN, y es el FORTRAN 77 *completo* con todas sus reglas y posibilidades; el otro recibe el nombre de subconjunto FORTRAN que denominaremos en adelante *básico* y es una parte del primero, pero suficiente para desarrollar aplicaciones.

El contenido de este libro está basado en el FORTRAN 77 completo y con él pueden codificarse multitud de aplicaciones que luego pueden ser ejecutadas por una amplia variedad de máquinas.

Codificar es una parte de la tarea más general que es la programación. Codificar es aquel mecanismo mediante el cual un determinado algoritmo se convierte, siguiendo unas reglas o códigos, en un programa capaz de ser ejecutado por un ordenador. En este libro aprenderemos a codificar aplicaciones usando las reglas del lenguaje FORTRAN en su versión más moderna.

1.2. CONCEPTOS BASICOS

A continuación se definirán algunos conceptos usuales en Proceso de Datos y, por otra parte, básicos para el desarrollo de este texto, algunos de los cuales ya han sido esbozados anteriormente.

Un sistema de cálculo u *ordenador*, está constituido esencialmente por los elementos siguientes (Fig. 1.1):

- Unidad de proceso, comúnmente conocida como *Unidad Central de Proceso*. Esta unidad es el corazón de la máquina, de manera que constituye el centro de trabajo y control del sistema de proceso de datos. Está formada por un gran número de componentes

electrónicos. Pueden considerarse en ella tres partes bien diferenciadas. La *unidad aritmética y lógica,* encargada de realizar operaciones como la suma, resta, multiplicación y división. También contiene circuitos para efectuar operaciones lógicas, como comparar dos datos y deducir cuál de ellos es el mayor, etc. La *Unidad de Control,* que interpreta las instrucciones de un programa y coordina todos los dispositivos restantes del sistema. Las *Unidades de Memoria,* encargadas de almacenar instrucciones, datos y resultados de un proceso. Existen memorias de alta velocidad, denominadas *Memoria Principal,* contenidas normalmente en la Unidad Central de Proceso, y otras de más baja velocidad, aunque mucho más económicas, que reciben el nombre de *Memoria Secundaria:* discos y cintas magnéticas, diskettes, etc.

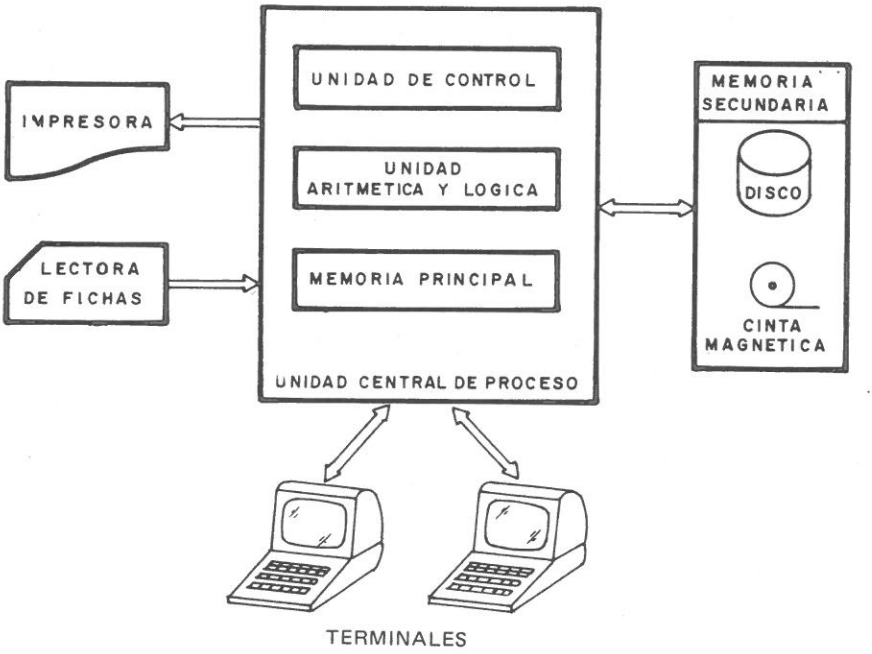


Figura 1.1

– Unidades de *Entrada/Salida,* como las impresoras, las antiguas lectoras de fichas perforadas y los terminales en general. Son las encargadas de transmitir al sistema de cálculo, los datos, los pro-

gramas o de recibir los resultados obtenidos después de finalizar un proceso.

Las memorias principales son de tres clases, dependiendo de la tecnología empleada para su construcción:

- *Memoria de Núcleos,* formada por material magnético en forma de anillo o toroide.

- *Memoria de Semiconductores* o memoria *RAM* (Random Access Memory) formada por circuitos electrónicos microscópicos capaces de representar dos estados: 0 y 1. El acceso a esta memoria es más rápido. Ella es más compacta y más barata de fabricar que la anterior.
  Las memorias de núcleos y de semiconductores son volátiles, lo cual significa que pierden la información que contienen cuando no reciben energía eléctrica.

- *Memoria ROM* (Read Only Memory) es una alternativa de la memoria RAM, en la cual la información reside en ella de forma permanente desde su fabricación, aunque no exista suministro de energía eléctrica.

Una *instrucción de máquina* puede compararse con una tecla de función de una calculadora de bolsillo, con la cual se suman dos números, por ejemplo, se dividen o almacena un dato en memoria. Todas las instrucciones capaces de ser realizadas por un ordenador reciben el nombre de *conjunto* o *juego de instrucciones.*

Empleando instrucciones adecuadas del conjunto de instrucciones, se describen las aplicaciones para que puedan ser ejecutadas por la máquina, es decir, se escriben los *programas de ordenador.* Los programas están formados por una secuencia lógica de instrucciones, es decir, las instrucciones se ordenan de manera que su conjunto responda a y desarrolle un algoritmo determinado.

Los programas de ordenador eran escritos en un principio en el propio *lenguaje de máquina* del que ya se ha hablado anteriormente. Hoy día los programadores utilizan lenguajes de programación más sencillos de aprender y usar, como el FORTRAN, pero que deben de ser traducidos a lenguaje de máquina por el propio ordenador mediante programas especiales suministrados normalmente por el fabricante y que se denominan *compiladores* o *intérpretes.* Ambos conceptos no son completamente sinónimos.

Cuando se utiliza un lenguaje de alto nivel como el FORTRAN, el proceso de *compilación* consiste básicamente en lo siguiente: convertir las instrucciones FORTRAN en una serie de instrucciones de máquina equivalentes, antes de proceder a ejecutar el programa. Cada instrucción FORTRAN puede dar como resultado de la compilación una o varias instrucciones de máquina.

El programa escrito en lenguaje FORTRAN recibe el nombre de *programa fuente* y el programa obtenido después de la compilación, *programa objeto.*

Los compiladores se suministran, cuando el ordenador es pequeño, en cinta de papel, cinta magnética, diskettes, o cinta de cassette. Con frecuencia, los compiladores para microordenadores están permanentemente contenidos en la memoria ROM. Para los grandes sistemas, residen en disco magnético.

Durante el proceso de compilación se obtienen *mensajes de diagnóstico,* lo cual significa que cada instrucción escrita en FORTRAN es analizada para comprobar si cumple con todas las reglas sintácticas del lenguaje y, por consiguiente, si es o no convertible en instrucción de máquina. Cuando no ocurre así, se producen *errores sintácticos* que el programador debe corregir, procediendo luego a una nueva compilación. El programa ya correcto se almacena para ser ejecutado, a la vez que se obtiene un listado del mismo.

Aunque no existan errores sintácticos, puede ocurrir que el programa posea *errores lógicos,* es decir, errores del propio diseño del programa. Tales errores no pueden ser detectados por el compilador y únicamente se descubren cuando el programa objeto se procesa y se obtienen resultados inesperados o inadecuados.

Básicamente, el proceso de compilación podría esquematizarse según el diagrama de la figura 1.2.

1.3. CARACTERES, CONSTANTES Y VARIABLES

El lenguaje FORTRAN, como cualquier otro lenguaje de programación, posee unos elementos básicos que se estudiarán a continuación y un conjunto de reglas que, combinando entre si esos elementos de base, van dando lugar a una serie de instrucciones, con las cuales el hombre llega a comunicarse con el ordenador, de manera que éste ejecutará correcta y rápidamente el algoritmo diseñado por el programador. Veamos cuáles son esos elementos básicos.

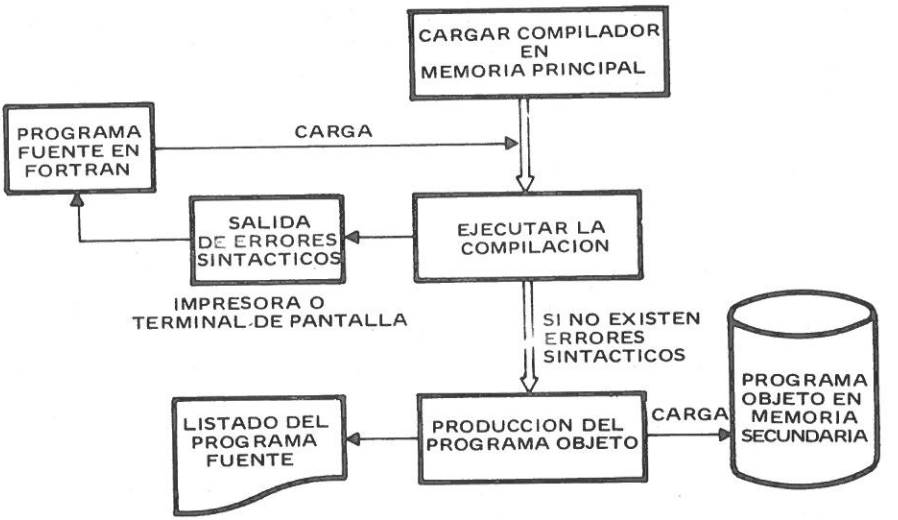


Figura 1.2

1.3.1. Conjunto de caracteres FORTRAN

El lenguaje FORTRAN dispone de un *conjunto de caracteres*, formado por veintiséis letras (mayúsculas), diez dígitos y once caracteres especiales. Estos son los siguientes:

a) Caracteres alfabéticos (*):

A B C D E F G H I J K L M N O P Q R S T
U V W X Y Z

b) Caracteres numéricos:

0 1 2 3 4 5 6 7 8 9

c) Caracteres especiales (Ver cuadro página siguiente).

El conjunto de los caracteres alfabéticos y numéricos forman el conjunto de los *caracteres alfanuméricos.*

(*) Algunos sistemas permiten el uso de letras mayúsculas y minúsculas.

Carácter	Nombre
	Blanco
=	Signo igual
+	Signo más
−	Signo menos
*	Asterisco
/	Slash
(	Paréntesis de apertura
)	Paréntesis de cierre
,	coma
.	Punto
'	Apóstrofe

El caracter *blanco* algunas veces se representa mediante la letra minúscula b, pero únicamente a efectos de que la redacción de un programa resulte más clara y cómoda. En cualquier caso, no tiene significado.

Se ha dicho anteriormente que FORTRAN 77 posee dos versiones: dependiendo de cuál de ellas se utilice, el conjunto de caracteres especiales puede ser de once en el FORTRAN 77 básico o de trece en el FORTRAN 77 completo. Este incluye como adicionales, los siguientes: los dos puntos (:) y el signo de dólar ($).

En cuanto a ordenación se refiere, la secuencia para los caracteres alfabéticos es la indicada anteriormente, es decir, A es menor que B, D es menor que Z, etc. Para los numéricos, 1 es menor que 2, etc. No existe una secuencia de ordenación para los especiales que pueda considerarse como general. (Ver apartado 7.4, pág. 292).

Actualmente y para mayor claridad, algunas veces, la letra O se distingue del dígito cero 0, acompañando éste de un slash, así: Ø.

1.3.2. Constantes FORTRAN

El término *constante* designa un valor específico y determinado que se define al hacer un programa y que no cambia a lo largo del mismo.

Son constantes, por ejemplo, los valores:

3,14159 y 2,71828

que corresponden respectivamente a π y e.

Existen cuatro tipos de constantes en el FORTRAN 77 básico y seis en el completo. Estos son los siguientes:

Básico	Completo
Enteras	Enteras
Reales	Reales
Lógicas	Lógicas
Carácter	Carácter
	Doble precisión
	Complejas

Más adelante se definirá cada una de las que corresponden al FORTRAN 77 básico, así como las de doble precisión del completo. Por no ser demasiado usuales, el estudio de las complejas se hará aparte.

1.3.3. Variables FORTRAN

El valor de la longitud de una circunferencia de radio r es $2\pi r$. En esta expresión matemática existen dos constantes, el **2** y **3,14159**, y una variable que es el radio r.

El concepto de *variable* para el lenguaje FORTRAN coincide con el concepto habitual que se tiene de ella: es un nombre simbólico con el que se designa o hace referencia a un dato que puede tomar valores diversos.

En FORTRAN, las variables se forman empleando los caracteres alfanuméricos estudiados. Más concretamente: una *variable* FORTRAN es una agrupación de caracteres alfanuméricos, unos a continuación de otros, de manera que el primero de ellos sea alfabético.

Son variables FORTRAN las siguientes:

```
A
ABX
RADIO
A13
```

He aquí algunas reglas a tener en cuenta al formar variables:

a) El número total de caracteres no debe exceder de seis. Esta es la variable standard, aunque algunos compiladores permitan variables más largas.

b) No están permitidos los caracteres especiales.

c) Es conveniente denominar las variables mediante caracteres que recuerden el concepto que representen. Por ejemplo:

```
RADIO,   para un radio
VELO,    para una velocidad
FUNC,    para una función
```

Veamos algunos ejemplos de variables FORTRAN no válidas:

```
3AB,        comienza por un carácter numérico
AbRE,       contiene un carácter especial blanco b
ACELERA,tiene más de seis caracteres
```

El significado de variable, cuando se trata de ordenadores, posee un matiz adicional al explicado anteriormente cuando se dio su definición. Una variable, además de indicar un dato (una distancia, una temperatura, un salario), que puede tomar distintos valores, hace referencia también a una determinada zona o localización de la memoria del ordenador donde se archiva o almacena el dato correspondiente.

Así, por ejemplo, RADIO indica no sólo el valor de un radio de circunferencia, sino también el nombre simbólico con el que tanto programador como ordenador reconocen el lugar, en memoria, donde se aloja ese valor numérico. La figura 1.3 indica, de una forma general, este concepto.

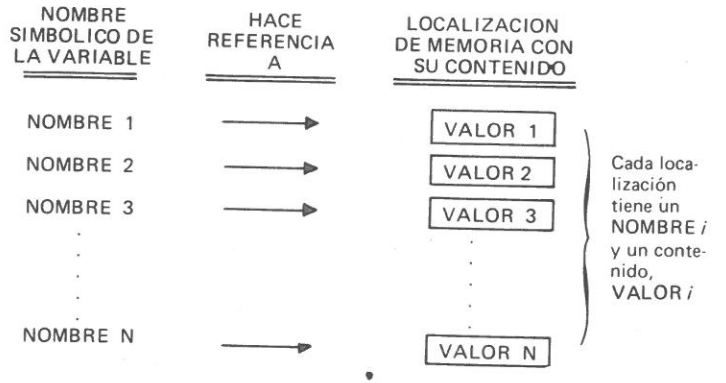


Figura 1.3

Puesto que las variables designan constantes numéricas y, como se ha dicho anteriormente, existen varios tipos de constantes en el FORTRAN, también existirán varios tipos de variables, los cuales serán estudiados más adelante. Estos tipos son:

Enteras
Reales
Lógicas
Caracter
Doble Precisión
Complejas

Por último, debe advertirse que es muy común designar tanto las constantes como las variables con el nombre genérico de *datos.*

1.4. ALGUNOS ASPECTOS DE LA MEMORIA PRINCIPAL

Para llegar a entender claramente cómo están almacenados en la memoria principal los datos, puede imaginarse la memoria como un conjunto de cajones o depósitos, denominado cada uno de ellos, *posición de memoria.* Estos depósitos o posiciones están numerados, para poderlos identificar y distinguir y cada uno de esos números identificadores recibe el nombre de *dirección de memoria.* La numeración comienza en cero y es creciente (Fig. 1.4).

Cuando una posición de memoria principal recibe un dato desde el exterior, el dato permanece en ella mientras otro dato no venga a sustituirlo y destruirlo. Cuando un dato se toma de una posición de memoria, para ser extraido al exterior por ejemplo, el dato no obstante sigue a su vez permaneciendo en la memoria. Lo mismo que sucede cuando una cinta magnética está reproduciendo música.

Cada posición de memoria está formada por un conjunto de diminutas parcelas capaces de recibir dos estados cada vez, ON y OFF. El estado ON equivale al 1, el estado OFF al 0. Cada una de esas parcelas, recibe el nombre de *bit* (*bi*nary digi*t* = dígito binario). De esta forma, cada posición de memoria permite contener la información codificada mediante una sucesión de ceros y unos. Pero veamos, en detalle, cómo están distribuidos esos bits dentro de cada posición.

Las posiciones de memoria se denominan también *palabras* y cada palabra se subdivide en otras unidades más pequeñas que reciben el nombre de *bytes.*

El ejemplo, cuyo diagrama se encuentra en la figura, es una de las posibilidades, ya que, dependiendo de la arquitectura del ordenador, esos esquemas pueden variar de unos a otros.

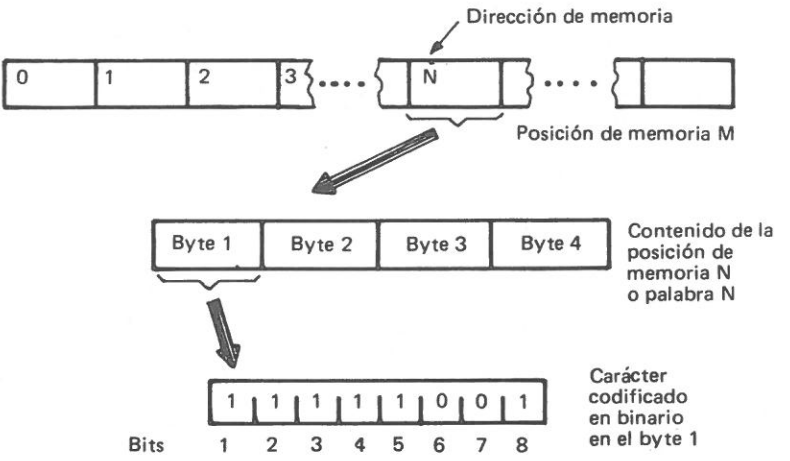


Figura 1.4

Cada byte está formado por bits a los que anteriormente se ha hecho mención. En nuestro ejemplo cada byte se compone de 8 bits y, porque en cada byte puede codificarse una letra, un número o un carácter especial, es por lo que a los bytes también se les designa con el nombre de *caracteres*. El caracter codificado en binario en el byte número 1 de la posición de memoria N de nuestro esquema es un 9 (=11111001) en código EBCDIC (Extended Binary Coded Decimal Interchange Code).

Algunos esquemas-tipo son los siguientes:

- La mayor parte de microordenadores utilizados en casa, colegios y escuelas son máquinas cuya palabra tiene un byte de longitud, es decir, se trata de micros de 8 bits.
- Otros microordenadores y miniordenadores son *micros de 16 bits* o *máquinas de 16 bits*, lo cual significa que cada palabra está compuesta por dos bytes de ocho bits cada uno.
- Ordenadores más grandes tienen incluso palabras de longitud mayor, como 24, 32, 36 ó 48 bits. El ejemplo mostrado en la figura anterior representaba un ordenador de palabras de 32 bits distribuidos en cuatro bytes de ocho bits.

Cada posición de memoria puede emplearse para archivar no sólo datos, sino también las instrucciones de que se compone un programa. Debe tenerse en cuenta que cada bit está constituido por nucleos magnéticos o celdas de semiconductores como ya se mencionó en "1.2 CONCEPTOS BASICOS".

El tamaño de la memoria se mide en K ($K = 2^{10} = 1024$). Por ejemplo, existen micros de 32K palabras de memoria de 8 bits, es decir, de 32K bytes; un mini de 256K palabras de 32 bits, tendrá una capacidad en bytes o caracteres de 4 x 256 x 1024 = 1048576.

Un múltiplo de K es M o *mega*. Así, 4 Mbyte significa 4 millones de bytes o caracteres.

Por último, ha de tenerse en cuenta que al programar utilizando como lenguaje el FORTRAN, se puede acceder al byte, pero no puede accederse a cada uno de los bits que lo componen. Byte es la mínima información a manejar.

1.5. CONSTANTES ENTERAS

Una constante entera es una sucesión de dígitos precedidos o no del signo positivo (+) o negativo (−) y sin coma decimal, es decir, el concepto de constante entera en FORTRAN coincide con el concepto habitual de número entero.

Si la constante es positiva, no es necesario escribir delante de ella el signo +.

Una constante entera puede ser nula.

Constantes enteras válidas serían:

45 −837 +31 Ø 15253 −7 ØØ12 25b3

La constante +31 puede también representarse por 31 y la ØØ12 por 12, en ambos casos con el mismo significado. La constante 25b3 equivale a la 2503 (cero y blanco son interpretados igual por el compilador).

Ejemplo de constantes enteras no válidas serían:

12,53	contiene una coma decimal
7S23	contiene un caracter alfabético (S)
75.237	contiene un punto

En FORTRAN se emplea el caracter especial punto (.) en lugar de la coma (,) para separar la parte entera de la fraccionaria de una cantidad, como más adelante se verá al tratar de las constantes reales. Basado en ésto y considerando que una constante entera, aunque en ella no figura el punto decimal es como si éste siempre estuviera situado en una localización fija a la derecha de todos sus dígitos, es por lo que las constantes enteras también se denominan *de punto fijo.*

Lo mismo que sucede con las calculadoras de bolsillo, donde no es posible representar cantidades enteras que sobrepasen un número determinado de dígitos, también utilizando FORTRAN existen limitaciones en cuanto a la magnitud de las cantidades enteras con las que vaya a trabajarse. La explicación es así: los números enteros, como cualquier otro dato, ya sabemos que se almacenan en la memoria de la máquina en forma binaria (unos y ceros) y en un total de bits que depende del ordenador de que se trate. Así, por ejemplo, si consideramos que una constante entera ocupa una palabra de memoria de 32 bits y uno de esos bits se emplea para colocar el signo positivo o negativo, quedan libres 31. En estas condiciones el número entero mayor es:

$$2^{31} - 1 = 2\ 147\ 433\ 647$$

y el más pequeño

$$-2\ 147\ 433\ 648$$

es decir, la precisión (número total de dígitos de una cantidad) sería aproximadamente de 9 ó 10 cifras.

He aquí algunos límites característicos para constantes enteras:

Máquinas de 16 bits por palabra: -32768 y $+32767$

Otros límites: -2^{31} y $2^{31}-1$,

-2^{35} y $2^{35}-1$,

Deben consultarse, en cada caso, las posibilidades del sistema que se utilice para no cometer errores de exceso de capacidad (positiva o negativa) al manejar constantes enteras en FORTRAN.

1.6. CONSTANTES REALES

Existen tres clases de constantes reales:

a) Constante real básica.

b) Constante real básica con exponente

c) Constante entera con exponente.

En cualquier caso, una constante real en FORTRAN equivale al valor de un número real habitual, es decir, una cantidad formada por una parte entera y una fraccionaria (punto decimal equivale a coma).

a) Constantes reales básicas

La forma de una *constante real básica* es: un signo (opcional), seguido de una serie de dígitos con un punto delante de todos ellos, entre dos de ellos o al final.

Ejemplos válidos:

+ 37.855 (= 37,855)	123.75
− 0.0021	+ .4573 (= 0,4573)
397.432	− .0044 (= −0,0044)
44. (= 44,0)	1814.0
.33 (= 0,33)	

b) Constante real básica con exponente

Se define como *exponente real* en FORTRAN, el carácter alfabético E seguido opcionalmente por un signo + o − y por una constante entera formada por dos dígitos como máximo:

E 27 , E+42 , E−3 , E−07 , E−33 , E5

son ejemplos válidos de exponentes reales. El ejemplo anterior E+42 podría haberse escrito E42 y el E5 como E+5 ó como E+05.

El significado de un exponente real es el equivalente a una potencia de diez. Así:

E25	equivale a	10^{25}
E−3	equivale a	10^{-3}
E+42	equivale a	10^{42}

Dada esta definición previa, diremos que una *constante real básica con exponente* es una constante real básica seguida de un exponente real y significa el producto de la constante que precede a la E por la potencia de diez que indica el exponente real.

Ejemplos válidos	Significado
5.2703E0	$5{,}2703 \times 10^{0}$
−3.287E−7	$-3{,}287 \times 10^{-7}$
0.043E30	$0{,}043 \times 10^{30}$
+8.753E−01	$8{,}753 \times 10^{-1}$
−.734E+04	$-0{,}734 \times 10^{4}$
325.E−1	$325{,}0 \times 10^{-1}$

Ejemplos inválidos serían:

1.E112	El exponente real contiene más de dos dígitos.
2.3E	Al exponente le falta su valor
+ 21.3E2.3	El exponente no está formado por una constante entera.

La constante real que precede a E suele denominarse también *mantisa,* mientras que la constante entera que sigue a la E se conoce también con el nombre de *característica.*

c) Constante entera con exponente

Se define como tal una constante entera seguida de un exponente real. Tienen el mismo significado que las constantes reales básicas con exponente.

Ejemplos válidos:

+1528E–03	$(= 1528{,}0 \times 10^{-3})$
–13E2	$(= -13{,}0 \times 10^{2})$
142573E99	$(= 142573{,}0 \times 10^{99})$
23E00	$(= 23{,}0 \times 10^{0} = 23{,}0)$

Las constantes reales que se han definido hasta aquí, se conocen también con el nombre de constantes de *punto flotante* ya que la posición del punto decimal depende realmente de cuál sea el valor del exponente o característica.

Por ejemplo:

$37.21E01 \equiv 37{,}21 \times 10^{1} = 372{,}1$
$37.21E{-}2 \equiv 37{,}21 \times 10^{-2} = 0{,}3721$
$37.21E5 \equiv 37{,}21 \times 10^{5} = 3721000{,}0$
$37E2 \equiv 37{,}0 \times 10^{2} = 3700{,}0$

Cuando una constante real tiene situado su punto decimal precediendo a todos los dígitos significativos de la mantisa, se dice que está *normalizada* o *tipificada.*

Ejemplos de constantes reales normalizadas son:

+.43257E−7 −.234807E02 .257336E5

Lo mismo que ocurría con las constantes enteras, también las constantes reales están sometidas a dos limitaciones:

- El número total de dígitos que constituyen el número y que pueden ser almacenados en el ordenador: es la precisión.
- Su magnitud (valor absoluto) que debe estar comprendida entre unos ciertos límites.

Si se sobrepasan, bien la precisión o la magnitud, la constante real será inválida porque no podrá almacenarse en memoria con el valor requerido. Estas dos condiciones dependen del ordenador con el que se trabaje.

En la figura 1.5 se muestra el rango de unas constantes reales muy usuales. Este rango equivale aproximadamente a una precisión de 7 dígitos decimales.

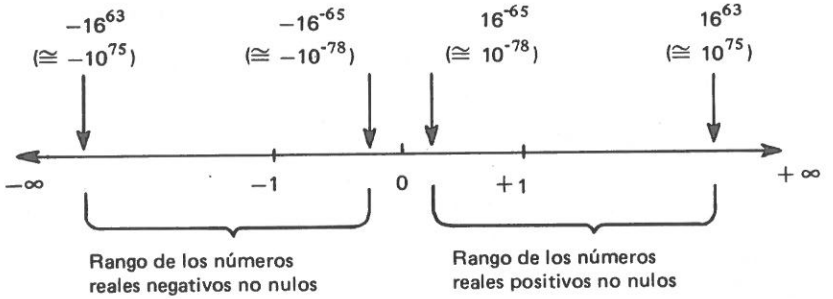


Figura 1.5

En este caso, una constante real válida sería cualquiera comprendida en los intervalos $(-16^{63}, -16^{-65})$, $(16^{-65}, 16^{63})$ y el cero incluido.

Otros ejemplos son:

$(-1{,}7 \times 10^{38}, -3{,}0 \times 10^{-39})$, $(3{,}0 \times 10^{-39}, 1{,}7 \times 10^{38})$ y el cero

$(-10^{322}, -10^{-294})$ $(10^{294}, 10^{322})$ y el cero.

1.7. CONSTANTES LOGICAS

Un *dato lógico* únicamente posee dos valores: *cierto* o *falso*. Estos datos también pueden almacenarse en la memoria del ordenador mediante códigos binarios especiales.

Las formas y valores de las *constantes lógicas* FORTRAN son:

FORMA	VALOR
.TRUE.	CIERTO
.FALSE.	FALSO

Obsérvense los puntos (.) que preceden y siguen a estas constantes.

1.8. CONSTANTES CARACTER

Un *dato carácter* es un conjunto de caracteres FORTRAN válidos y su *longitud* es el número total de caracteres que contiene. Pues bien, se denomina *constante carácter* a un conjunto de uno o más caracteres FORTRAN precedidos y seguidos por el carácter especial apóstrofe ('). Estos caracteres se denominan en este caso *delimitadores*.

Por ejemplo, son constantes carácter, las siguientes:

```
'LIBRO DE TEXTO'
'SALARIO MENSUAL='
```

La primera tiene una longitud de 14 caracteres, ya que en este caso los dos blancos entre las tres palabras son significativos. La segunda es de 16 caracteres de longitud.

Cuando sea necesario escribir un apóstrofe en medio de una constante carácter, éste vendrá representado por dos apóstrofes. Por ejemplo, para indicar:

```
EL VALOR 'MEDIO' ES
```

como constante caracter, se escribirá así:

```
'EL VALOR ''MEDIO'' ES'
```

que tiene 21 caracteres de longitud (existen tres blancos entre las cuatro palabras).

Es necesario observar ciertas limitaciones en cuanto al número de caracteres que pueden almacenarse en una palabra de memoria: mientras unos ordenadores (32 bits por palabra) admiten 4, otros llegan hasta 10. Otro límite es el número máximo de caracteres que puede contener una constante carácter y que depende del compilador utilizado.

No confundir dos apóstrofes con el doble apóstrofe que se encuentra en algunos teclados de terminales (comillas).

1.9. CONSTANTES DOBLE PRECISION

Estas constantes no están consideradas en el FORTRAN 77 básico y sí en el completo. Dada su importancia y uso, vamos a definirlas a continuación.

Se denomina *exponente doble precisión* en FORTRAN, al carácter alfabético D seguido opcionalmente de un signo + o − y finalizando por una constante entera formada por dos dígitos como máximo:

D27, D+03, D−3, D4

son ejemplos válidos. D27 significa 10^{27} ; D+03, 10^{3} ; D−3, 10^{-3}, etc.

Basado en este definición, existen dos clases de constantes doble precisión: *constantes reales básicas* con exponente *doble precisión* y *constantes enteras* con exponente *doble precisión*. La similitud con las constantes reales, previamente definidas, es total. La diferencia entre unas y otras es que el compilador reserva más memoria para las doble precisión, con lo que el número de dígitos disponibles es mayor (precisión).

a) *Constantes reales básicas* con exponente *doble precisión*.

Ejemplos válidos de estas constantes, serían:

245.3D−03	equivale a	$245{,}3 \times 10^{-3}$
−.24537D+3	equivale a	$-.024537 \times 10^{3}$ y está normalizada
342.00D+2	equivale a	$342{,}00 \times 10^{2}$

b) *Constantes enteras* con exponente *doble precisión*

Ejemplos válidos, serían:

+1528D−5	equivale a	$1528{,}0 \times 10^{-5}$
−13D2	equivale a	$-.13{,}0 \times 10^{2}$
123D00	equivale a	$123{,}0 \times 10^{0} = 123{,}0$

1.10. VARIABLES ENTERAS Y REALES

Una variable FORTRAN cuyo nombre comience con los caracteres alfabéticos I, J, K, L, M o N se dice que queda *implícitamente* definida como *variable entera.* Las variables enteras designan zonas de memoria cuyo contenido es una constante o valor entero.

Todas aquellas variables cuyo primer carácter alfabético sea una letra distinta de las mencionadas anteriormente, se dice que son *variables reales* definidas de forma *implícita.* Una variable real contiene un valor constante real.

Ejemplos:

VARIABLES ENTERAS	VARIABLES REALES
I	X
J2	A7
KA3	DISTA
MINIMO	XMAXIM
KG	SUB2

1.11. VARIABLES LOGICAS

Son datos que contienen una constante lógica. Las variables *lógicas* han de definirse o declararse, como tales, al principio de un programa FORTRAN, utilizando la palabra (sentencia de tipo) LOGICAL. Por ejemplo, la variable LUZ es entera según la definición dada anteriormente por comenzar su nombre con una L. Pero si se desea que únicamente tome los valores *cierto* o *falso,* debe definirse como una variable lógica así:

```
LOGICAL LUZ
```

Otros ejemplos son:

```
LOGICAL  KMS
LOGICAL  VELOCI
```

Las tres variables LUZ, KMS y VELOCI podrían declararse como *lógicas* empleando una sola definición y separándolas mediante comas:

```
LOGICAL  LUZ, KMS, VELOCI
```

1.12. VARIABLES CARACTER

Una *variable carácter* contiene un conjunto de caracteres FORTRAN y, como en el caso de las lógicas, ha de definirse o declararse como tal al principio del programa utilizando la palabra o instrucción CHARACTER y el número total de caracteres que va a ocupar dicha variable en la memoria del ordenador. Este concepto se denomina *especificación de longitud*. Por ejemplo:

```
CHARACTER*10 TITULO, KMS
```

significa que se están definiendo dos variables, TITULO y KMS, como variables *carácter* y que cada una de ellas tendrá *diez* caracteres FORTRAN de longitud.

Si las variables tienen distintas longitudes, es necesario escribir varias instrucciones CHARACTER. Así:

```
CHARACTER*5  X, Y
CHARACTER*10 TITULO
CHARACTER*4  I
```

definen las variables X, Y de cinco caracteres, TITULO de diez e I de cuatro.

Las tres instrucciones anteriores también podrían haberse escrito así:

```
CHARACTER*5  X, Y, TITULO*10, I*4
```

Cuando no se especifica longitud en CHARACTER se supone que es uno. Así:

```
CHARACTER VELOCI, ACELE, TITULO*4
```

define tres variables carácter: VELOCI y ACELE de un solo carácter de longitud y TITULO de cuatro.

Obsérvese que al definir una variable como *lógica* o *carácter* ya no importa cuál sea su primer carácter alfabético. No obstante la longitud máxima de su nombre debe ser de seis carácteres, como cualquier variable.

La especificación de longitud viene indicada por un asterisco * y una constante entera sin signo y no nula. No tendría sentido indicar

como longitud, en caracteres, un valor tal como 4.21, escribiendo, por ejemplo:

```
CHARACTER*4.21  TITULO
```

1.13. VARIABLES DOBLE PRECISION

En correspondencia con las constantes de doble precisión, también pueden definirse variables que contengan valores con más dígitos que los permitidos en una constante real normal: éstas son las *variables de doble precisión,* las cuales se definen al principio del programa mediante la instrucción DOUBLE PRECISION. Así:

```
DOUBLE  PRECISION  META, BASE, X
```

proporciona al compilador la información correspondiente para que considere las variables META, BASE y X como variables con una precisión (número de dígitos) más amplia que las variables reales normales.

Las variables de *doble precisión* únicamente pueden emplearse cuando se dispone de un FORTRAN 77 completo.

Las tres últimas instrucciones que hemos estudiado:

```
LOGICAL
CHARACTER
DOUBLE  PRECISION
```

se denominan instrucciones o *sentencias de tipo*. Existen más instrucciones de esta categoría que estudiaremos más adelante; pero lo que es necesario entender desde ahora, es que esta clase de instrucción FORTRAN, sirve para cambiar, el tipo de una variable cualquiera que de por sí (implícitamente) sería entera o real según comenzara o no por los caracteres I, J, K, L, M o N.

1.14. CUADRO RESUMEN

Como resumen de lo tratado hasta aquí en relación con los distintos datos FORTRAN, constantes y variables, se adjunta el cuadro siguiente, donde puede verse, de forma comparativa, cada uno de los conceptos estudiados, acompañados de varios ejemplos.

		DATOS FORTRAN	
		CONSTANTES	VARIABLES
ENTERAS O DE PUNTO FIJO		+3; −455; 8213	METRO; I325; LONGI; KILO; J
REALES O DE PUNTO FLOTANTE	BASICAS	8.75; −43.720; .453; −357	COORDE ; X23 ; A POBLA ; BASICO
	REALES CON EXPONENTE	Sin normalizar: 43.2E02; −7.0E−1 Normalizadas: .3745E+4; .37E−5	
	ENTERAS CON EXPONENTE	43252E−07; −37403E+02	
LOGICAS		. TRUE . ; . FALSE .	LOGICAL METRO LOGICAL X3,I53,COOR
CARACTER		'TITULO=' ; 'X E Y'	CHARACTER*4 LUZ, CIMA CHARACTER*5 I, J, MU*2, X*6
DOBLE PRECISION (*)	REALES CON EXPONENTE	Sin normalizar: 43.2D02; −7.0D−1 Normalizadas: .3745D+04; .375D−5	DOUBLE PRECISION HACHE
	ENTERAS CON EXPONENTE	43252D−07 ; −3704D+2	DOUBLE PRECISION X3, VEL, B

Cuadro 1.1.

(*) Unicamente en FORTRAN 77 completo. En el presente cuadro-resumen quedan separadas de las comunes por la doble línea.

1.15. LA HOJA DE CODIFICACION FORTRAN

Cuando fue diseñado el lenguaje FORTRAN en los años cincuenta, el principal y casi único medio de transmitirle al ordenador la información, es decir, los programas y los datos, era utilizando la ficha perforada.(Fig. 1.6) donde cada carácter se codificaba de acuerdo con unas determinadas perforaciones realizadas con máquinas especiales (perforadoras). La ficha perforada, aún utilizada actualmente, tiene ochenta columnas; como consecuencia, en ella pueden perforarse ochenta caracteres, alfabéticos, numéricos o especiales.

Aunque actualmente se trabaja en esta dirección, por razones económicas o de sencillez, aún no se dispone de medios que puedan leer directamente la información preparada de forma escrita para el ordenador o que puedan interpretar y transmitir la voz del hombre.

Por ello, los programas o se perforan en fichas que son luego interpretadas por la máquina mediante unidades de entrada electromecánicas o bien se emplean otros medios como son: la cinta perforada de papel, la cinta magnética o el más común: el terminal visual. Este terminal consta de un teclado semejante al de una máquina de escribir, donde se encuentra el juego de caracteres además de otras teclas para usos específicos, y un visor o pantalla de televisión, donde puede visualizarse la información que se está preparando para el ordenador. Los caracteres tecleados o bien se van almacenando directamente en la memoria del ordenador o en una memoria intermedia, cinta magnética, disco magnético, diskette, de donde pasarán en su momento y, una vez acabado el programa, a la memoria principal para ser sometidos a

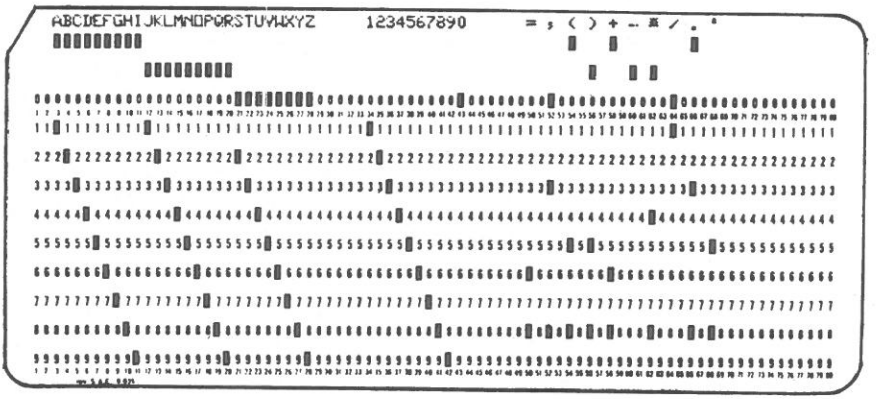


Figura 1.6

HOJA DE CODIFICACION FORTRAN

HOJA_____DE_____

PROGRAMA	SISTEMA
PROGRAMADOR	FECHA

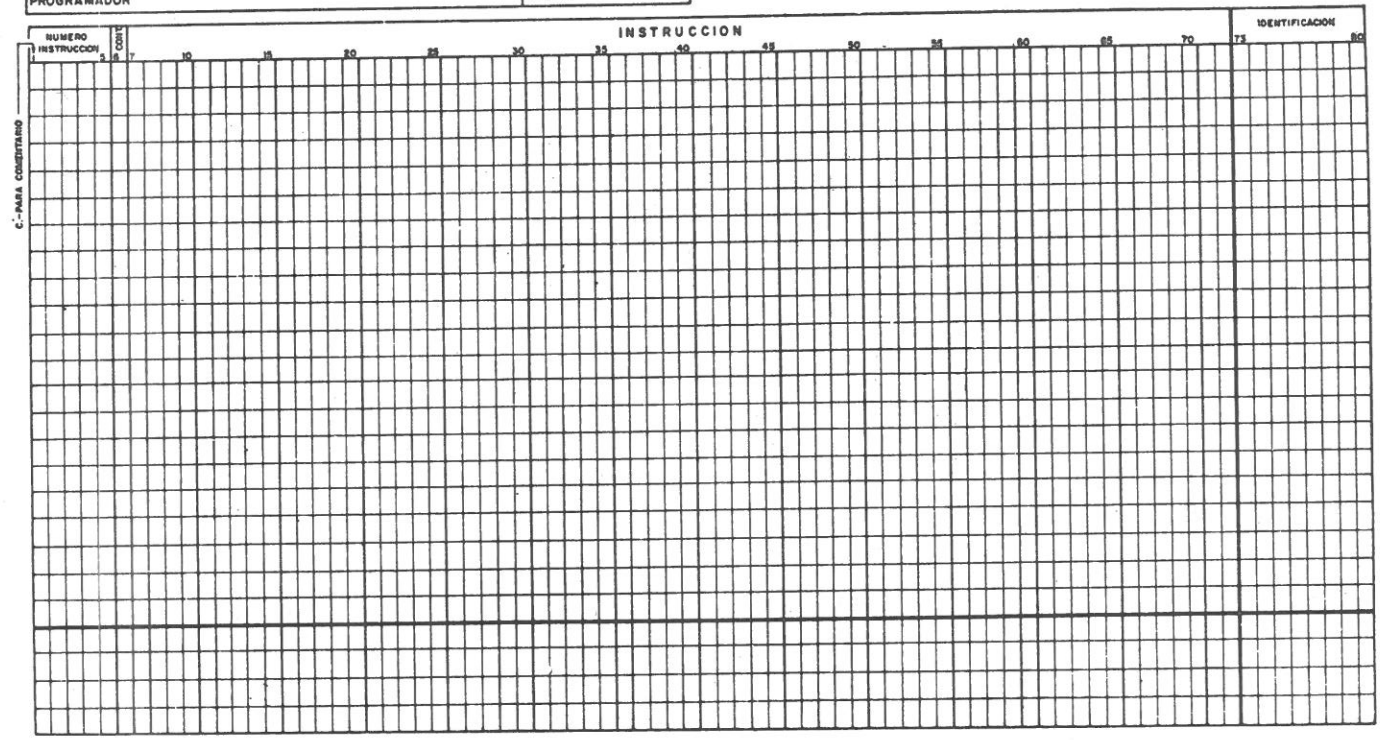


Figura 1.7

proceso: en nuestro caso, a compilación, por ejemplo, si se trata de la preparación de un programa fuente FORTRAN.

El lenguaje FORTRAN fue diseñado, como ya se ha dicho, para perforar sus instrucciones en tarjetas de 80 columnas. Aunque hoy día éste no sea el medio más utilizado, la estandarización del FORTRAN 77 sigue conservando esta filosofía y, por ello, existen unas hojas, denominadas *hojas de codificación FORTRAN* donde se escribe el programa que más tarde se introducirá en el ordenador para ser compilado. Estas hojas tienen la apariencia mostrada en la figura 1.7, y todo lo contenido en ellas es pasado luego a fichas (cada línea en una ficha distinta) o bien introducido al ordenador, como antes se ha mencionado, empleando terminales teclado-pantalla, por ejemplo.

A continuación se va a describir el uso de cada una de las ochenta columnas de que se compone esta hoja matriz.

Se denomina *línea* de un programa FORTRAN a una sucesión de 72 caracteres tomados del conjunto de caracteres FORTRAN 77. Cada carácter ocupa o se escribe en una *columna* y éstas se numeran consecutivamente 1, 2 hasta 72, de izquierda a derecha. Las líneas se escriben u ordenan en la misma secuencia en que se van a introducir en el ordenador para ser compiladas y luego ejecutadas.

Línea de comentario es aquella línea que contiene en la columna 1, una C o un asterisco * y a continuación y a partir de la columna 2, el comentario que el programador desee incluir en su programa. Un comentario es una información descriptiva que puede ayudar a leer y comprender el programa que se está escribiendo. Los comentarios no son traducidos por el compilador, pero son incluidos en los listados que se obtienen de los programas fuente. Véase figura 1.8.

Figura 1.8

Línea inicial es cualquier línea que no sea de comentario y que contenga en la columna 6, o un cero o un blanco. Figura 1.9.

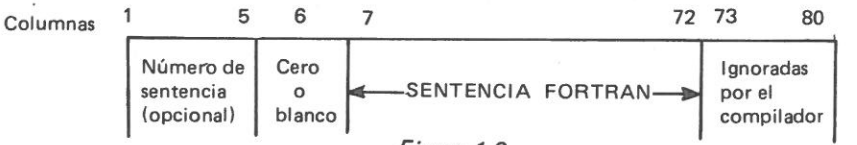


Figura 1.9

Las columnas 1 a 5 pueden contener el número o etiqueta de la sentencia FORTRAN que viene a continuación en la misma línea.

El *número* o *etiqueta* de una sentencia está formado por una constante entera de uno a cinco dígitos, escritos en cualquier posición dentro de las columnas 1 a la 5. No todas las sentencias tienen que llevar número; solamente aquellas a las que se haga referencia desde otra en un mismo programa. A dos sentencias o instrucciones distintas no se les puede asignar el mismo número.

En cualquiera de las columnas 7 a la 72 se escribe, en toda línea inicial, la propia sentencia FORTRAN. Más adelante estudiaremos todas esas sentencias y sus reglas sintácticas.

Cada sentencia puede escribirse con los caracteres unos a continuación de otros o bien dejando blancos entre ellos para una mejor y cómoda lectura en los listados.

La mayor parte de las sentencias podrán escribirse en una sola línea inicial; pero si éste no fuera el caso, podrían emplearse una o varias líneas de continuación. En todo caso, el número máximo de caracteres, incluidos los blancos, que puede contener una sentencia FORTRAN es de 660 en el básico y 1320 en el completo.

Una *línea de continuación* (fig. 1.10) es una línea que no es de comentario y que contiene cualquier carácter FORTRAN, distinto del cero o del blanco en la columna 6. El número máximo de líneas de continuación que puede tener una línea inicial es de 9 en el FORTRAN 77 básico y de 19 en el completo.

Figura 1.10

Los caracteres de la columna 6 pueden ser números o letras consecutivos para ordenar de esa manera las líneas de continuación.

Las columnas 1 a 5 deben contener blancos y las 7 a 72 la continuación de una sentencia FORTRAN.

Téngase en cuenta que aunque la sentencia sea muy corta, no está permitido escribir en una misma línea varias sentencias.

Las columnas 73 a 80 se denominan columnas de *identificación* y no son analizadas por el compilador, aunque la información que con-

tengan será reproducida en los listados. Estas columnas pueden utilizarse para identificar el programa con un nombre, para numerar las líneas o para cualquier otro propósito.

1.16. LAS SENTENCIAS FORTRAN

Para acabar este capítulo, vamos a dar a continuación una serie de conceptos relacionados, todos ellos, con las sentencias FORTRAN. Este capítulo es como una introducción y todos los conceptos que en él se están estudiando constituyen la base de capítulos siguientes.

Una *sentencia* FORTRAN o instrucción, es una secuencia de caracteres válidos que siguen unas reglas sintácticas y que tienen una semántica o significado para el programador y el compilador. Como ya hemos dicho, una instrucción FORTRAN se convierte en general en varias instrucciones de máquina.

Las sentencias se clasifican en *ejecutables* y *no ejecutables*: las primeras especifican acciones, como "leer un dato", "imprimir un resultado", "hacer una comparación", etc.; las segundas describen características, como "asignar valores iniciales a una variable", "forma de imprimir un resultado", "declarar o definir el tipo de una variable", etc.

Las sentencias FORTRAN se forman mediante *palabras clave* a base de una secuencia específica de letras que tienen un significado para el lenguaje FORTRAN. Ejemplos de estas palabras clave son: IF, CONTINUE, GO TO, DO, etc. Serán estudiadas a lo largo del texto.

En FORTRAN, se denomina *unidad programa,* al conjunto formado por una serie de sentencias y, opcionalmente, líneas de comentario. Una unidad programa puede ser o bien un *programa principal* o un *subprograma*. Estos conceptos serán definidos más tarde.

En el cuadro que se da a continuación figuran todas las sentencias del FORTRAN 77 básico y del completo.

Una entrada en el cuadro marcada con una X o con un – indica, respectivamente, que la sentencia correspondiente existe o no en ese FORTRAN. En los capítulos que siguen, estudiaremos todas las sentencias del FORTRAN 77 completo.

Se conocen con el nombre genérico de *sentencias de especificación* las siguientes: DIMENSION, COMMON, EQUIVALENCE, IMPLICIT, EXTERNAL, INTRINSIC, SAVE y PARAMETER.

Las sentencias INTEGER, REAL, LOGICAL, CHARACTER, DOUBLE PRECISION y COMPLEX también son de especificación,

	SENTENCIAS FORTRAN 77	Básico	Completo
EJECUTABLES	Aritméticas, lógicas, ASSIGN, asignación carácter	X	X
	GO TO incondicional, GO TO asignado, GO TO calculado	X	X
	IF aritmético, IF lógico	X	X
	CONTINUE	X	X
	STOP, PAUSE	X	X
	DO	X	X
	READ, WRITE	X	X
	PRINT	–	X
	REWIND, BACKSPACE, ENDFILE, OPEN	X	X
	CLOSE, INQUIRE	–	X
	CALL, RETURN	X	X
	END	X	X
NO EJECUTABLES	PROGRAM, FUNCTION, SUBROUTINE	X	X
	ENTRY, BLOCK DATA	–	X
	DIMENSION, COMMON, EQUIVALENCE, IMPLICIT, EXTERNAL, INTRINSIC, SAVE	X	X
	PARAMETER	–	X
	INTEGER, REAL, LOGICAL, CHARACTER	X	X
	DOUBLE PRECISION, COMPLEX	–	X
	DATA	X	X
	FORMAT	X	X
	Función de sentencia	X	X

Cuadro 1.2

pero más comúnmente se conocen con el nombre general de *sentencias de tipo.*

De acuerdo con estas definiciones y considerando el FORTRAN 77 completo, el orden en que deben escribirse las sentencias y las líneas de comentario, es el indicado en el cuadro siguiente:

<table>
<tr><td rowspan="5">Líneas de Comentario</td><td colspan="3">Sentencias PROGRAM, FUNCTION, SUBROUTINE o BLOCK DATA</td></tr>
<tr><td rowspan="4">Sentencias FORMAT y ENTRY</td><td rowspan="2">Sentencias PARAMETER</td><td>Sentencias IMPLICIT</td></tr>
<tr><td>Otras sentencias, de especificación</td></tr>
<tr><td rowspan="2">Sentencias DATA</td><td>Sentencias Función Sentencia</td></tr>
<tr><td>Sentencias Ejecutables</td></tr>
<tr><td colspan="3">Sentencia END</td></tr>
</table>

Cuadro 1.3

Las líneas verticales indican clases de sentencias que pueden entremezclarse. Así, un FORMAT puede colocarse entre las IMPLICIT o entre las Funciones de Sentencia. Las líneas horizontales indican aquellas clases de sentencias que no pueden mezclarse: las Funciones de Sentencia, por ejemplo, no pueden escribirse y colocarse entre las sentencias ejecutables.

Por último, las palabras utilizadas para formar las sentencias FORTRAN, no pueden emplearse como nombres de variables.

EJERCICIOS RESUELTOS

1.1. Decir si son o no válidas las siguientes constantes enteras. Cuando no sean válidas, explicar por qué.

a) 57,347
b) 32701
c) −25.0
d) 341
e) 273−
f) 1729
g) 43(2
h) −7241
i) 15b8

Respuesta

Las constantes b), d), f), h) e i) son correctas. La constante 15b8, donde b significa blanco, es correcta, ya que equivale a 1508.

a) Contiene una coma
c) Contiene un punto
e) Tiene el signo menos (−) al final
g) Contiene un paréntesis

1.2. Escribir las siguientes cantidades como constantes enteras FORTRAN:

a) +57347
b) −8707.0
c) $3{,}83 \times 10^2$
d) −12
e) $4700{,}00 \times 10^{-2}$
f) 343,69
g) $-10{,}852 \times 10^1$

Respuesta

a) 57347 ó +57347
b) −8707
c) 383
d) −12
e) 47
f) No es posible. Unicamente podría escribirse su parte entera 343
g) No es posible. Unicamente podría escribirse −108 (su parte entera).

1.3. A continuación se da una lista de constantes reales válidas y no válidas. Decir a qué clase pertenecen, dando las razones por las que no son válidas.

a) −997.78
b) +0.
c) 34,71.8
d) 437
e) 1.E112
f) 2E−03
g) .473
h) 2.3E
i) −0.000411
j) 4)232
k) 21.3E2.3
l) 12.731E+02
m) E+03

Respuesta

a) Válida
b) Válida
c) Inválida. Contiene una coma.
d) Inválida. Le falta el punto decimal.
e) Inválida. El exponente tiene más de dos dígitos.
f) Válida.
g) Válida.
h) Inválida. Falta el valor del exponente.
i) Válida
j) Inválida. Contiene un paréntesis y le falta el punto decimal.
k) Inválida. El exponente no es una constante entera.
l) Válida.
m) Inválida. Un exponente no puede figurar solo.

1.4. Se da una serie de constantes expresadas en modo mantisa, potencia de diez. Expresarlas en forma de constantes reales FORTRAN y normalizar cada una de ellas cuando sea necesario.

a) $-2{,}33 \times 10^{-3}$
b) $0{,}043 \times 10^{3}$
c) $+8{,}730 \times 10^{2}$
d) $-0{,}734 \times 10^{4}$
e) $8{,}753 \times 10^{-1}$
f) $567{,}27 \times 10^{-5}$
g) $23{,}36 \times 10$
h) 2306,36

Respuesta

a) −2.33E−3 ≡ ≡ −.233E−2
b) .043E3 ≡ ≡ .43E2
c) +8.730E2 ≡ ≡ .8730E3
d) −.734E+4 ya está normalizada.
e) 8.753E−1 ≡ ≡ .8753E0
f) 567.27E−5 ≡ ≡ .56727E−2
g) 23.36E01 ≡ ≡ .2336E3
h) 2306.36E0 ≡ ≡ .230636E04

1.5. Escribir las siguientes constantes, como constantes reales y como constantes doble precisión, ambas con exponente:

a) 1523
b) −17,243
c) 0,000157
d) $57{,}41 \times 10^{-8}$
e) 31×10^{3}
f) −124127
g) 7120,83
h) $37{,}437 \times 10^{-3}$

Respuesta

a) 1523.E0 ó 1.523E3 " 1523.D0 ó .1523D4
b) −17.243E0 ó −.17243E2 " −17.243D0 ó −.17243D2
c) 0.000157E0 ó .157E−3 " 0.000157D0 ó .157D−3
d) 57.41E−8 ó .5741E−6 " 57.41D−8 ó .5741D−6
e) 31.E03 ó .31E05 " 31.D03 ó .31D05
f) −124127.E0 ó −.124127E6 " −124127.D0 ó −.124127D6
g) 7120.83E0 ó .712083E4 " 7120.83D0 ó .712083D4
h) 37.437E−3 ó .37437E−1 " 37.437D−3 ó .37437D−1

1.6 Las siguientes son constantes doble precisión. Decir cuáles son válidas y cuáles no lo son, explicando por qué. (El blanco viene inclinado por b).

a) 217D03
b) 1.447D+09
c) −3483D−121
d) 4bb3.2D−7
e) 44.03E12
f) 4*83.D−7
g) −.1321D1.8
h) 230636.

a), b), d) Son válidas. 4bb3.2D−7 equivale a 4003.2D−7
c) Inválida. El exponente tiene más de dos dígitos.
e) Inválida. El exponente no es un exponente de doble precisión.
f) Inválida. Contiene un símbolo especial (*)
g) Inválida. El exponente no es una constante entera
h) Inválida. Falta el exponente de doble precisión

1.7. Escribir como constantes carácter, los siguientes títulos, indicando el número total de caracteres que tendrá la constante carácter FORTRAN (b indica blanco) correspondiente.

a) LA b MEDIA b ES =
b) COORDENADA bb X bb =
c) EL b 'MIEDO' b ES b LIBRE
d) CALCULO bb DE bb VALORES
e) 'GRAN b SESION'
f) PTS = bbb F.F. = bbb

Respuesta

a) 'LA b MEDIA b ES = ' Total de caracteres = 12
b) 'COORDENADA bb X bb = ' Total de caracteres = 16
c) 'EL b "MIEDO" b ES b LIBRE' Total de caracteres = 19
d) 'CALCULO bb DE bb VALORES' Total de caracteres = 20
e) '"GRAN b SESION"' Total de caracteres = 13
f) 'PTS = bbb F.F. = bbb' Total de caracteres = 15

1.8. Cuáles de los nombres siguientes son válidos y cuáles no, como variables enteras FORTRAN, definidas implícitamente.

a) NIDO
b) A123
c) I520
d) MTS
e) KILOGRAMO
f) LOMA
g) J3+2
h) N(5)L
i) VELOCI

Respuesta

a), c), d) y f) son válidas
b) Inválida. No comienza por un carácter I, J, K, L, M, N
e) Inválida. Tiene más de seis caracteres.
g) Inválida. Contiene un carácter especial +
h) Inválida. Contiene dos caracteres especiales (y)
i) Inválida. No comienza por un carácter I, J, K, L, M, N

1.9. Clasificar los nombres siguientes en variables FORTRAN válidas o inválidas y las válidas en enteras o reales.

a) AKMS
b) JOTAS
c) PARAMETRO
d) STOP
e) RAYO
f) L124

Respuesta

a) Real válida
b) Entera válida
c) Inválida. Más de seis caracteres.
d) Inválida. Es palabra reservada al FORTRAN
e) Real válida
f) Entera válida

2

Sentencias básicas

La vida probablemente es redonda

VAN GOGH *(1853-1890)*

2.1. OPERADORES ARITMETICOS

En Matemáticas se dispone de una serie de símbolos que indican determinadas operaciones a efectuar con los datos: Así por ejemplo, el + para la suma, el – para la resta, etc.

De la misma forma, el lenguaje FORTRAN posee también esos signos, tomados del conjunto de caracteres especiales, con los cuales se indican las operaciones aritméticas. Esos signos se denominan *operadores aritméticos*, y son los cinco indicados en el cuadro siguiente:

OPERADOR	SIGNIFICADO
* *	Exponenciación
/	División
*	Multiplicación
–	Resta o negación
+	Suma

Cuadro 2.1

Así, las expresiones algebraicas que siguen a continuación, se escribirían en FORTRAN como se indica:

a + b	A + B
a • b	A * B
$\frac{c}{d}$	C/D
a^x	A ** X
–m	–M

Cada uno de los símbolos **, /, * siempre operan sobre un par de operandos y por eso se escriben entre ellos. Los operadores + y – pueden operar sobre dos operandos, escribiéndose en este caso entre ellos (primer ejemplo de los citados) o bien sobre uno sólo, en cuyo caso se colocan delante del mismo (último ejemplo).

2.2. EXPRESIONES ARITMETICAS

En general en FORTRAN, una *expresión aritmética* se forma utilizando operandos, es decir, constantes y/o variables, operadores y paréntesis, de manera que agrupados indiquen la ejecución de un determinado cálculo aritmético. Según ésto, una sola constante o una sola variable también son expresiones aritméticas FORTRAN: las más simples de todas.

Así la fórmula que indica el cálculo de la longitud de una circunferencia, $2\pi r$, podría indicarse en FORTRAN mediante la expresión aritmética:

```
2 * PI * RADIO
```

constituida por *una constante entera,* 2, *dos variables reales,* PI *y* RADIO, *y un operador,* *, *repetido dos veces.*

Valor de una expresión es el que ésta toma sustituyendo las variables por los valores que les correspondan y efectuando todas las operaciones que en ella se indiquen. Si en el ejemplo anterior fuera PI = 3,14 y RADIO = 2, se obtendría:

```
2 * 3.14 * 2 = 12.56
```

Ejemplos de expresiones artiméticas son los siguientes:

EXPRESION ALGEBRAICA	EXPRESION FORTRAN
$\frac{a+b}{c}$	(A + B) / C
$\frac{1}{2} gt^2$	0.5* G * T ** 2
$a\,(b-c)$	A * (B – C)

EXPRESION ALGEBRAICA	EXPRESION FORTRAN
$\frac{3a}{x+y}$	(3*A) / (X + Y)
$5a^3$	5 * A ** 3
2,8 (t + 3)	2.8 * (T + 3)
$b^2 - 4ac$	B ** 2 – 4 * A * C
$ax^2 + bx + c$	A * X ** 2 + B * X + C

Puede suceder que todos los datos, constantes y variables, de una expresión FORTRAN sean enteros, en cuyo caso se dice que la *expresión* es *entera.* Si todos los datos son reales, la *expresión* es *real.* Si existen de ambos tipos, real y entero, la *expresión* es *mixta.* Doble precisión se considera como real.

Por ejemplo:

3 * I + M	es entera
(2.5 + A) * 3.8	es real
(8.75 / 2.5) ** 2.2	es real
2 * (3 + VELO)	es mixta
RESTO + 4	es mixta

El cuadro que sigue indica cómo es el resultado obtenido al combinar mediante los operadores +, –, *, / dos datos cualesquiera enteros, reales, doble precisión o una mezcla de dos de ellos:

+, –, *, /	DATO ENTERO	DATO REAL	DATO D. PRECISION
DATO ENTERO	RESULTADO ENTERO	RESULTADO REAL	RESULTADO D. PRECISION
DATO REAL	RESULTADO REAL	RESULTADO REAL	RESULTADO D. PRECISION
DATO D. PRECISION	RESULTADO D. PRECISION	RESULTADO D. PRECISION	RESULTADO D. PRECISION

Cuadro 2.2

Cuando los datos se combinan entre sí utilizando la exponenciación **, el cuadro que contiene el tipo de resultado obtenido, sería:

**		EXPONENTE		
		DATO ENTERO	DATO REAL	DATO D. PRECISION
BASE	DATO ENTERO	RESULTADO ENTERO	RESULTADO REAL	RESULTADO D. PRECISION
	DATO REAL	RESULTADO REAL	RESULTADO REAL	RESULTADO D. PRECISION
	DATO D. PRECISION	RESULTADO D. PRECISION	RESULTADO D. PRECISION	RESULTADO D. PRECISION

Cuadro 2.3

Ejemplos aclaratorios de estos cuadros y referidos al operador /, son los que se indican a continuación. Supóngase que las variables A, B y C, que son reales, tienen como valores:

A = 8.0
B = 4.0
C = 3.0

Sean I, J y K tres variables enteras, tales que:

I = 8
J = 4
K = 3

Entonces las expresiones aritméticas FORTRAN que se indican, producirían los resultados siguientes:

EXPRESION	VALOR		EXPRESION	VALOR	
A/B	2.0	(real)	J/A	0.5	(real)
I/J	2	(entero)	B/C	1.333...	(real)
A/C	2.666...	(real)	J/K	1	(entero)
J/I	0	(entero)	J/C	1.333...	(real)

En los ejemplos cuarto y séptimo, el resultado ha sido entero, pero *truncado,* es decir, se han perdido todas las cifras correspondientes a la parte fraccionaria.

La formación de expresiones aritméticas FORTRAN está sometida a una serie de reglas que vamos a enunciar y explicar con ejemplos.

a) No está permitido escribir dos operadores adyacentes; entre ellos debe existir un paréntesis. Así, los ejemplos dados en la primera columna no están de acuerdo con esta regla. Son válidos los correspondientes de la segunda columna:

A+-B	A+(-B)
A*-B	A*(-B)
-A-+B	-A-(+B)
C--D	C-(-D)
E**-K	E**(-K)

b) En una expresión FORTRAN, como sucede en Matemáticas, por cada paréntesis de apertura debe existir otro de cierre.

La siguiente expresión es correcta:

```
((A + B) * 3) / 2.8
```

e indica la fórmula algebraica:

$$\frac{3\ (a + b)}{2{,}8}$$

Sin embargo, no es válida la siguiente:

```
((A + B * 3) / 2.8
```

por faltar un paréntesis de cierre.

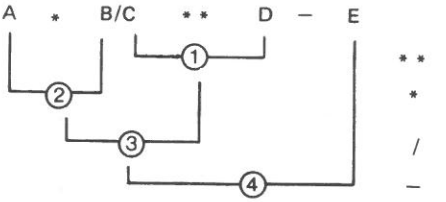
c) Cuando en una expresión FORTRAN no se hace uso de paréntesis, por no ser imprescindibles para separar las operaciones que desean indicarse, la *prioridad* con que éstas se realizan es, de mayor a menor, la siguiente:

Potencias
Multiplicación y división
Suma y resta

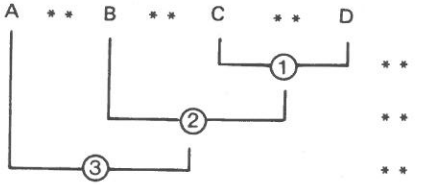
Obsérvese, además, lo siguiente: cuando en una misma expresión se indican operaciones de la misma prioridad, multiplicaciones y divisiones o sumas y restas, estas se ejecutan recorriendo la expresión de *izquierda a derecha;* cuando en una misma expresión lo que existe son varias potencias consecutivas, se ejecutan de *derecha a izquierda.*

A continuación se proponen varios ejemplos, sin el uso del paréntesis, mostrandose la forma en que se realizan, etapa por etapa. Los círculos numerados indican el orden de ejecución y los signos al margen la propia operación.

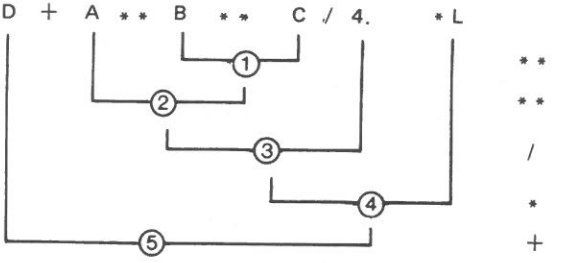
1. A * B/C * * D–E



2. A * * B * * C * * D



3. D + A * * B * * C/4. * L

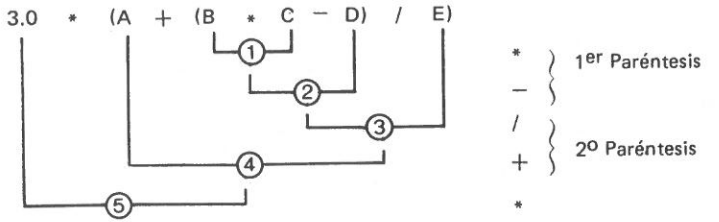


d) Cuando en una expresión aritmética FORTRAN se hace uso de paréntesis, las operaciones se ejecutan siguiendo las prioridades estudiadas en la regla anterior, dentro de cada par de paréntesis, y desde los paréntesis más interiores a los más exteriores.

Así, la expresión:

3.0 * (A + (B * C – D) / E)

donde se marcan los paréntesis emparejados, se realizaría, por etapas, siguiendo el orden que se indica:



2.3. CONJUNTO DE DATOS

En Proceso de Datos existe un concepto importante, denominado lista. Por ejemplo, un conjunto de notas de clase, de precios de distintos artículos, puede disponerse convenientemente en secuencia de elementos, cada uno de los cuales puede ser identificado según el lugar que ocupa. Así el conjunto de las notas de una clase de diez alumnos podría darse de este modo:

NOTAS = (8, 10, 1, 5, 6, 9, 3, 3, 7, 5)

Lo mismo sucedería con una *tabla* o matriz de doble entrada donde en cada posición se encontrara el número total de piezas rectangulares registradas en una muestra y clasificadas según su largo y su ancho.

Listas y tablas son estructuras de datos que vamos a designar con el nombre genérico de conjunto de datos (en inglés: array).

Se denomina *conjunto de datos* o simplemente conjunto, a una agrupación de datos, constantes o variables, todos ellos del mismo ti-

po, enteros o reales, entre los cuales existe una cierta relación de homogeneidad: notas en el primer ejemplo; total de piezas en el segundo. El primer ejemplo constituye un caso de *conjunto lineal* o de una sola dimensión; el segundo es de *dos dimensiones* (depende de dos características).

En FORTRAN, todo conjunto se reconoce por un nombre, que es una variable. Así, el conjunto de notas de clase, podría denominarse genéricamente por:

```
NOTAS
```

y el de piezas rectangulares por:

```
PIEREC
```

Cada dato del conjunto se denomina en FORTRAN *elemento* del mismo. Así, el conjunto NOTAS tenía diez elementos y cada uno de ellos posee una localización o situación dentro del conjunto. Por ejemplo, el cuarto elemento de NOTAS se indica en FORTRAN por:

```
NOTAS (4)
```

y su contenido es 5. Del mismo modo NOTAS (8) vale 3, y es el octavo elemento, etc. y de una forma general:

```
NOTAS (I)
```

indica el elemento i-esimo del conjunto de una sola dimensión NOTAS.

Como se ha dicho anteriormente, un conjunto puede poseer más de una dimensión. Supongamos que la tabla de piezas rectangulares, PIEREC, citada anteriormente, fuera la siguiente:

	10	20	30	40
2	80	25	50	0
4	1	3	24	83
6	0	4	94	101

significando que: el número total de piezas de 10 x 2 es 80; de 20 x 2 es 25; de 40 x 6 es 101, etc.

Para localizar ahora cada elemento de esta tabla o matriz, se emplean dos números: uno que indica la fila del elemento y otro la columna. La tabla anterior tiene tres filas y cuatro columnas. Así, el elemento que ocupa la segunda fila, tercera columna, vale 24 y se indica en FORTRAN

```
PIEREC (2,3)
```

es decir, los elementos de un conjunto de dos dimensiones se indican dando el nombre de la variable que designa el conjunto y *dos* números encerrados entre paréntesis y separados por una coma: el primero representa la fila y el segundo la columna donde se encuentra dicho elemento.

A continuación se indican los valores de algunos de los elementos de la tabla PIEREC.

```
PIEREC (2,3) = 24
PIEREC (3,3) = 94
PIEREC (3,4) = 101
PIEREC (1,3) = 50
```

El elemento PIEREC (5,2), por ejemplo, no existe. Un elemento cualquiera, que ocupe la fila *m* y la columna *n*, podría indicarse así:

```
PIEREC (M, N)
```

Los números encerrados entre paréntesis y que indican la posición de cada elemento dentro del conjunto, se denominan *subíndices*. Cada elemento del conjunto de las notas poseía un solo subíndice; los de piezas, dos. Cada elemento de un conjunto de tres dimensiones, tendrá tres subíndices y así sucesivamente.

A cada conjunto utilizado en FORTRAN, le corresponde un tipo implícitamente, es decir, los conjuntos cuyo nombre genérico es una variable entera, estarán constituidos por elementos que son todos valores enteros; aquellos cuyo nombre es una variable real, poseen elementos que son todos reales. Los dos conjuntos NOTAS y PIEREC poseerían, respectivamente, elementos enteros y reales.

Toda variable FORTRAN acompañada de subíndices entre paréntesis, se conoce con el nombre de *variable con subíndices* o *variable subindicada.*

Veamos algunos ejemplos más:

INTER (2,8)	Indica un elemento del conjunto INTER, de dos dimensiones, constituido por elementos enteros.
NUMERO (10, 3, 1)	Es un elemento del conjunto tridimensional NUMERO de datos enteros.
POLAR (8)	Octavo elemento del conjunto real lineal POLAR
CIMA (4, 5, 7, 3)	Elemento del conjunto de cuatro dimensiones CIMA, formado por elementos todos reales.

2.4. FORMACION DE SUBINDICES

Existen algunas reglas y limitaciones a tener en cuenta al escribir subíndices. Como ya sabemos, el número de dimensiones de un conjunto coincide con el número de subíndices que posee cada variable del mismo. Pues bien; el número mínimo de dimensiones en ambos FORTRAN es de *uno;* el máximo es de *tres* en el FORTRAN básico y de *siete* en el completo.

Cada *subíndice* tiene que ser siempre de *tipo entero*, es decir, puede ser una constante entera, una variable entera o bien una *expresión aritmética* pero *entera.*

Por ejemplo, una variable subindicada podría ser ANUL (J). Según el valor que tome en cada caso la variable J, así será el elemento apuntado del conjunto real ANUL.

Algunos ejemplos correctos de variables subindicadas serían:

```
FOCO (3)
NUM (I, 3*J)
PIEREC (4 + I, 2, 3*J)
LUN (M/5)
```

Ejemplos incorrectos son:

ALEC (3., J)	Contiene un subíndice real (3.).
X (4, 5. + A)	Contiene un subíndice que es una expresión real (5. + A).
Y (A, B, C)	Los subíndices son todos reales.
LUN (2 + I, 3 + B)	El segundo subíndice es una expresión mixta (3 + B)

2.5. DECLARACION DE CONJUNTOS

Con anterioridad a que un ordenador procese los datos de un programa, debe conocer qué datos son variables normales o simples y cuáles son elementos de un conjunto, entre otras cosas, porque debe reservar memoria para todos ellos. Por ejemplo, en el caso del conjunto anterior PIEREC debería reservar posiciones de memoria suficientes para almacenar 3 x 4 = 12 datos distintos. En el caso NOTAS, debería reservar memoria para diez elementos.

Ello se logra utilizando una instrucción FORTRAN no ejecutable de especificación con la que se declara el conjunto. Esa instrucción es:

DIMENSION

Por ejemplo, el conjunto NOTAS, debería declararse así:

```
DIMENSION NOTAS (1:10)
```

El conjunto de dos dimensiones PIEREC, así:

```
DIMENSION PIEREC (1:3, 1:4)
```

Entre los paréntesis que siguen al nombre del conjunto debe haber tantos pares de valores separados por comas (,) como dimensiones tenga el conjunto. Los valores de cada par se denominan, respectivamente, *límite inferior* y *superior* y van separados por dos puntos (:).

Así, NOTAS será un conjunto lineal o de una sola dimensión que posee un solo subíndice con valor mínimo de 1 y máximo de 10.

PIEREC es un conjunto bidimensional tal que el primer subíndice puede tomar valores entre 1 y 3 y el segundo entre 1 y 4.

Según esto, las variables subíndicadas siguientes no serían válidas:

NOTAS (11)	Se supera el límite superior.
NOTAS (0)	El subíndice es menor que el límite inferior
PIEREC (0,3)	El primer subíndice está por debajo del rango
PIEREC (3,8)	El segundo subíndice supera al límite superior.
PIEREC (4,5)	Ambos subíndices están fuera de su rango.

Las dos declaraciones anteriores pueden resumirse en una sola, así:

```
DIMENSION NOTAS (1:10), PIEREC (1:3, 1:4)
```

Otro ejemplo:

```
DIMENSION X (1:10), Y (1:20), Z (1:10, 1:15)
```

declara tres conjuntos de elementos reales: X, Y y Z. Los dos primeros son lineales con un máximo de elementos de 10 y 20, respectivamente. El tercero es bidimensional, con un máximo de 10 filas y 15 columnas.

El límite inferior tiene que ser menor que el superior, pero no tiene por qué ser 1; puede valer 0 ó tomar un valor negativo. Cuando el límite inferior sea 1, este valor puede omitirse.

Ejemplos:

```
DIMENSION  A (10),  B (10),  C (10,15)
DIMENSION  A (2:10), B (3:30), C (10, 2:10, 2:15)
DIMENSION  X (0:20)
```

El primer ejemplo equivaldría a escribir:

```
DIMENSION  A (1:10), B (1:10), C (1:10, 1:15)
```

En FORTRAN básico, el límite inferior tiene que ser siempre 1 y tiene que omitirse. Ejemplos válidos:

```
DIMENSION  A (20), X (3,5)
DIMENSION  X (20,20), Y (30), Z (3,4)
```

Mediante la sentencia DIMENSION, el compilador reconocerá cuánta memoria deberá reservar para el total de elementos que componen uno o varios conjuntos. Por ello, dicha sentencia debe aparecer en un programa FORTRAN al principio de este (véase cuadro 1.3).

2.6. DISPOSICION DE LOS CONJUNTOS EN MEMORIA

Veamos ahora cómo distribuye el compilador la memoria para colocar en ella un conjunto cualquiera.

a) Conjuntos unidimensionales

Sea A el nombre de un conjunto y supongamos la sentencia:

```
DIMENSION A (1:5)
```

o la equivalente:

```
DIMENSION A (5)
```

las cuales declaran un conjunto de elementos, todos ellos reales

```
A (1), A (2), A (3), A (4), A (5)
```

En la memoria del ordenador, dicho conjunto se distribuye así, a partir de una posición *n* cualquiera:

	POSICION DE MEMORIA	CONTENIDO
CONJUNTO A	n	A(1)
	n + 1	A(2)
	n + 2	A(3)
	n + 3	A(4)
	n + 4	A(5)

es decir, en orden creciente de posiciones de memoria, comenzando en una determinada *n*.

Entonces, el lugar que ocupa, dentro de un conjunto unidimensional, un elemento cualquiera cuyo subíndice sea I, es el

```
I-ésimo
```

b) Conjuntos bidimensionales

Sea un conjunto bidimensional B de dos filas y tres columnas declarado mediante una cualquiera de las dos sentencias equivalentes:

```
DIMENSION B (1:2, 1:3)
```

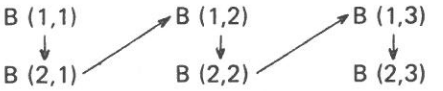
o la

```
DIMENSION B (2,3)
```

Este conjunto, para nosotros, estaría distribuido en una tabla como la que se indica:

B (1,1)	B (1,2)	B (1,3)
B (2,1)	B (2,2)	B (2,3)

Al pasar a la memoria del ordenador, los elementos anteriores se ordenan sucesivamente siguiendo el orden de las flechas:



es decir, por columnas. De manera que, en la memoria del ordenador, dicho conjunto se distribuiría a partir de una posición de memoria *n*, de la forma que se indica a continuación:

	POSICION DE MEMORIA	CONTENIDO	COLUMNA
CONJUNTO B	n	B(1,1)	1
	n + 1	B(2,1)	
	n + 2	B(1,2)	2
	n + 3	B(2,2)	
	n + 4	B(1,3)	3
	n + 5	B(2,3)	

Obsérvese como el segundo subíndice permanece invariable mientras que el primero va tomando todos los valores posibles: **1 y 2**.

Si representamos por M el límite superior de la primera dimensión del conjunto, dato que aparece en la propia sentencia DIMENSION, el lugar que ocupa un elemento cualquiera de subíndices I, J, sería el:

I + M*(J – 1)–ésimo

Por ejemplo, en el conjunto B anterior, en el que M = 2, el elemento de subíndices 1 y 3 (I = 1, J = 3) ocupa el lugar:

1 + 2 * (3–1) = 5

es decir, el quinto lugar en memoria, a partir de una posición determinada.

c) Conjuntos tridimensionales

Supongamos la sentencia

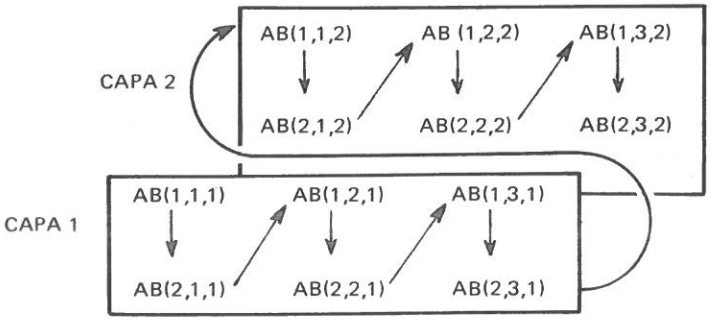
DIMENSION AB (1:2, 1:3, 1:2)

o la equivalente

DIMENSION AB (2, 3, 2)

en las que el primer subíndice indica el número máximo de filas, 2; el segundo el de columnas, 3; el tercero el de capas o páginas, 2.

Al pasar a la memoria, los elementos de AB se ordenarían sucesivamente siguiendo las flechas:



Como puede observarse, los elementos de estos conjuntos se almacenan en posiciones crecientes de memoria por páginas o capas y dentro de cada capa, como los bidimensionales, es decir, por columnas y a partir de una posición de memoria *n*, según se indica en el cuadro adjunto:

	POSICION DE MEMORIA	CONTENIDO	COLUMNA	CAPA
CONJUNTO AB	n	AB(1,1,1)	1	1
	n + 1	AB(2,1,1)		
	n + 2	AB(1,2,1)	2	
	n + 3	AB(2,2,1)		
	n + 4	AB(1,3,1)	3	
	n + 5	AB(2,3,1)		
	n + 6	AB(1,1,2)	1	2
	n + 7	AB(2,1,2)		
	n + 8	AB(1,2,1)	2	
	n + 9	AB(2,2,2)		
	n + 10	AB(1,3,2)	3	
	n + 11	AB(2,3,2)		

Obsérvese como mientras el tercer subíndice que indica la capa o página permanece fijo, los otros dos recorren todas sus posibilidades y siempre siguiendo la regla dada para los conjuntos bidimensionales.

Indicando por M el límite superior de la primera dimensión, por N el de la segunda (datos contenidos en la sentencia DIMENSION), un elemento cualquiera de AB de subíndices I, J, K, ocuparía el lugar

$$I + M * (J - 1) + M * N * (K - 1) - \text{ésimo}$$

Así, en el ejemplo anterior, el elemento AB(2, 1, 2) ocuparía el lugar:

$$2 + 2 * (1 - 1) + 2 * 3 * (2 - 1) = 8$$

es decir, el octavo a partir de una posición de memoria determinada.

Cuando se utiliza para programar el FORTRAN completo, todo lo dicho puede generalizarse a conjuntos con más de tres dimensiones.

2.7. SENTENCIA DE ASIGNACION ARITMETICA

Una vez examinados los conceptos fundamentales que constituyen la base del lenguaje FORTRAN, como son datos, expresiones, conjuntos, pasaremos a estudiar las instrucciones u órdenes mediante las cuales pueden programarse los distintos problemas.

La primera de estas sentencias, que pertenece al grupo de las sentencias ejecutables, es la *sentencia de asignación.*

Existen tres tipos distintos de sentencias de asignación:

Aritmética
Lógica
Carácter

pero las dos últimas serán estudiadas en otro capítulo.

Las sentencias de asignación *aritmética* sirven para ordenar al calculador la realización de cálculos aritméticos. Su forma general es

```
v = e
```

siendo: *v* el nombre de una variable con o sin subíndice de tipo entera, real o doble precisión. Esta última, únicamente en el FORTRAN completo.

e una expresión aritmética FORTRAN.

El signo "igual" no tiene aquí el mismo significado que en el cálculo matemático. En una sentencia de asignación aritmética, dicho signo quiere decir "es reemplazado por", es decir, "el valor de *v* es reemplazado por el obtenido para *e*".

Por ejemplo, la sentencia de asignación aritmética:

```
X = Y * Z
```

significa el cálculo del producto Y por Z (expresión real) y la sustitución del valor que tuviera X por el resultado de ese producto.

De la misma forma:

```
A (1) = B/C
```

significa que el valor que se obtenga para el cociente B/C reemplaza al valor que tuviera A(1), quedando almacenado, además, en dicha posición de memoria:

Otros ejemplos:

```
E = I * J + 4
```

I se multiplica por J y luego se le añade 4, sustituyendo ese resultado al valor de E.

```
F(3) = A + B
```

El valor resultado de la suma A + B, se coloca en el tercer elemento, F(3), del conjunto F.

```
A = A + B
```

Al valor de A se añade el de B y ese resultado sustituye de nuevo a A. Equivale a sumar al valor contenido en la posición de memoria A, el contenido en la posición de memoria B, colocando el resultado en la posición de memoria A, destruyéndose así el contenido que allí hubiera anteriormente.

```
I = I + 1
```

El valor de I se aumenta en una unidad y ese resultado se toma como nuevo valor de I. Constituye un *contador* que se va incrementando en una unidad cada vez que se utilice esa sentencia.

Las sentencias de asignación aritmética están sometidas a unas reglas que se encuentran en el cuadro siguiente, en el que las letras significan: E = Entero, R = Real, DP = Doble precisión (únicamente en el FOR-

TRAN completo).

TIPO DE v \ TIPO DE e	E	R	DP
E	E	E	E
R	R	R	R
DP	DP	DP	DP

Cuadro 2.4

Es decir, la ejecución de una sentencia de asignación aritmética causa la evaluación de la expresión *e* siguiendo las reglas dadas en el párrafo 2.2 (Cuadros 2.2 y 2.3) asignando dicho valor a *v* según las reglas del cuadro 2.4. Por ejemplo, cuando la expresión sea real, pero *v* sea entero, el resultado será entero; cuando la expresión sea doble precisión, pero *v* real, el resultado será real.

Ejemplos:

SENTENCIA	VARIABLES DE *e*	EXPRESION *e*	VALOR DE *v*
I = A * B	A = 3.2 (real) B = 1.5 (real)	A * B = 4.8 (real)	4 (entero)
A = I/J	I = 6 (entero) J = 2 (entero)	I/J = 3 (entero)	3.0 (real)
A = I/2 + 4.	I = 3 (entero)	I/2 + 4. = 5.0 (real)	5.0 (real)
M = B	B = 3.4 (real)	B = 3.4 (real)	3 (entero)
I = I + 2	I = 3 (entero)	I + 2 = 5 (entero)	5 (entero)

En los ejemplos primero y cuarto ha existido *truncamiento*, es decir, se ha perdido la parte fraccionaria del resultado al asignar *e* a *v*.

Una sentencia de asignación aritmética únicamente altera el valor de la variable del primer miembro *v*, dejando inalteradas las variables que figuren en la expresión FORTRAN *e* del segundo miembro.

A las variables de *e* se les ha debido adjudicar valores antes de aparecer en la sentencia de asignación aritmética. Así, en la sentencia:

```
SUEL = 2. * TO + 2.3 * GA ** 2 – V (I)
```

la única variable que se modifica es la SUEL y han de haberse definido en otro lugar del programa las variables TO, GA y V(I) antes de realizar el cálculo que dicha sentencia indica.

2.8. SENTENCIAS PROGRAM, STOP Y END

Al escribir un programa FORTRAN, su primera sentencia puede ser PROGRAM, cuya sintaxis es simplemente:

```
PROGRAM nombre
```

siendo *nombre*, el nombre o denominación simbólica que quiere darse al programa que comienza a escribirse. No es obligatorio utilizar esta sentencia; pero si figura en el programa, debe escribirse en primer lugar.

Para indicar la terminación de la ejecución de un programa FORTRAN es necesario emplear la sentencia STOP, cuya sintaxis o forma general es:

```
STOP n
```

donde *n* es una serie de hasta cinco dígitos o bien una *constante carácter.*

Ejemplos de sentencia STOP:

```
STOP
STOP 243
STOP 'FINAL'
```

Como veremos más adelante, un programa FORTRAN puede contener varias sentencias STOP o lo que es lo mismo, varios puntos de parada. Por ello, aunque el uso de *n* no es obligatorio, es conveniente para poder saber donde realmente se ha detenido un programa, ya que la constante *n* puede ser visualizada en una unidad de salida, cuando se produce la parada.

La última sentencia de un programa tiene que ser:

```
END
```

Esta sentencia equivale al final de un programa ejecutable e indica al compilador que ya no existen más sentencias en el programa fuente para ser convertidas en programa objeto.

Por resumir, diríamos que STOP produce una parada lógica y definitiva y sirve para detener un proceso; END sirve para indicar el final de una compilación.

La sentencia PROGRAM es una sentencia no ejecutable, mientras que STOP y END pertenecen a la categoría de las ejecutables.

Veamos ahora en un ejemplo muy sencillo, el uso de las sentencias estudiadas hasta ahora. Evidentemente, el programa que vamos a analizar a continuación, podrá mejorarse a medida que se estudien nuevas instrucciones.

Programa 1. Se trata de calcular el área de dos círculos cuyos radios respectivos son $r_1 = 2{,}3$ y $r_2 = 5{,}4$.

El programa podría ser:

```
C - PARA COMENTARIO    NUMERO INSTRUCCION    CONT    INSTR
                       1    5                6       7  10  15  20  25  30  35
                       PROGRAM CIRCULO
*
*  PROGRAMA PARA CALCULAR
*  EL AREA DE DOS CIRCULOS
*
                       DIMENSION RADIO(1:2),CIRCU(1:2)
                       PI=3.1415
                       RADIO(1)=2.3
                       RADIO(2)=5.4
                       I=1
                       CIRCU(I)=PI*RADIO(I)**2
                       I=I+1
                       CIRCU(I)=PI*RADIO(I)**2
                       STOP
                       END
```

Figura 2.1

He aquí algunos comentarios sobre el programa. Este comienza con la sentencia

```
PROGRAM CIRCULO
```

la cual indica que CIRCULO será la denominación simbólica o nombre atribuido al programa. La última sentencia física es END.

Existen cuatro líneas de comentario precedidas por el símbolo * que contienen la frase PROGRAMA PARA CALCULAR EL AREA DE DOS CIRCULOS como pauta para saber cuál es el objeto del programa. En lugar de * podría haberse escrito también C.

La sentencia:

```
DIMENSION RADIO (1:2), CIRCU (1:2)
```

equivalente a:

```
DIMENSION RADIO (2), CIRCU (2)
```

define dos conjuntos RADIO y CIRCU cada uno de ellos de una sola dimensión y con un máximo de dos elementos. Estos conjuntos son reales.

Las cuatro sentencias que siguen son sentencias de asignación aritmética. Asignan valores a las variables reales PI, RADIO(1), RADIO(2) y a la entera I. RADIO(1) y RADIO(2) son variables reales sub-indicadas.

La sentencia que sigue, realmente calcula:

```
CIRCU (1) = PI * RADIO (1) * * 2 = 3,1415 x 2,3²
```

es decir, el área del primer círculo.

Se ha escrito a continuación:

```
I = I + 1
```

que es otra sentencia de asignación aritmética, mediante la cual la variable I toma el nuevo valor:

```
I = I + 1 = 1 + 1 = 2
```

con el cual se calcula

CIRCU (2) = PI * RADIO (2) * * 2 = $3,1415 \times 5,4^2$

es decir, el área del segundo círculo. La variable I sirve de contador.

El programa CIRCULO se detiene al encontrar la sentencia STOP con lo cual finaliza.

Más adelante veremos otras formas de escribir este programa, todo ello a medida que dispongamos de más conocimientos sobre otras sentencias.

Las sentencias STOP y END pertenecen a una familia que se denomina sentencias de control.

2.9. SENTENCIA GO TO INCONDICIONAL

En los programas preparados para un ordenador, las instrucciones se ejecutan, mientras no se disponga lo contrario, secuencialmente, es decir, acabada la realización de una de ellas, comienza la ejecución de la siguiente. Esto también sucede empleando el lenguaje FORTRAN.

Existen sin embargo situaciones en las que no es conveniente continuar con la ejecución de la instrucción siguiente finalizada la realización de la precedente. Por el contrario; es necesario volver a repetir la ejecución de una instrucción anterior o producir un salto a otra sentencia de más adelante, ignorando la serie que queda entre ellas. Estos saltos se denominan *bifurcaciones* o transferencias de control.

Aquellas instrucciones que sirven para gobernar y dirigir la realización o ejecución de otras instrucciones, reciben el nombre genérico de *sentencias de control* y pertenecen a la familia de las ejecutables. Ya hemos estudiado dos de ellas, STOP y END y ahora estudiaremos otras.

Si la siguiente instrucción a ejecutar en un programa no es la que sigue en secuencia, es decir, si en un programa ha de realizarse una bifurcación, deberá indicarse de alguna forma qué instrucción es la siguiente a ejecutar, lo cual se consigue empleando la sentencia GO TO incondicional, por ejemplo.

La sintaxis de esta sentencia es:

```
GO TO n
```

siendo *n* el número o *etiqueta* de una sentencia ejecutable que aparecerá en el mismo programa: es, por consiguiente, una constante entera positiva y distinta de cero, de valor n $\leqslant$ 99999. Las palabras GO TO significan "IR A". Pueden escribirse también así: GOTO.

La constante *n* representa el número de la siguiente instrucción a realizar, es decir, el lugar preciso donde va a efectuarse la bifurcación.

Recordemos que una instrucción podía poseer un número que se escribía en las columnas 1 a la 5 de la hoja de codificación. Aquí encontramos una de sus aplicaciones.

Examinemos este ejemplo:

```
      A = 3 + VELO
      GO TO  321
      .
      .
      .
      .
      .
321   VELO = 2 * A
```

La primera sentencia es una instrucción de asignación aritmética y la segunda una bifurcación incondicional, es decir, un salto a la sentencia número 321, en la cual se duplica el valor que tuviera A, almacenándose en la variable o posición VELO. Los puntos en columna, significan cualquiera otra sentencia o conjunto de ellas, que en este caso se ignoran, es decir, no se ejecutan.

Las bifurcaciones pueden realizarse hacia adelante, como en el ejemplo o hacia atrás.

Otro ejemplo: supongamos que desea sumarse una serie de números enteros, comenzando en el 1, es decir, 1 + 2 + 3 + 4 + ...

Esto se lograría empleando la codificación:

```
      I = 0
      J = 1
10    I = I+J
      J = J+1
      GO TO 10
```

La sentencia número 10 va acumulando la suma de los enteros sucesivos. No obstante, esta codificación tiene un problema, y es que cons-

tituye un bucle sin fin: no sabemos, por ejemplo, como sumar sólo los 20 primeros enteros o los 200 primeros. Lo aprenderemos un poco más adelante.

El flujo lógico que seguiría la ejecución de un programa al encontrar una sentencia GO TO incondicional viene indicado mediante el esquema siguiente (fig. 2.2):

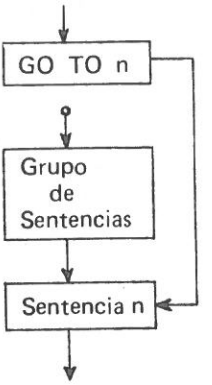


Figura 2.2

Como puede observarse "Grupo de Sentencias" nunca se ejecutará a menos que sea direccionado desde otro lugar del programa, es decir, que exista otra sentencia de bifurcación que señale un salto a dicho grupo.

2.10 SENTENCIA GO TO CALCULADA

Esta sentencia responde al esquema o sintaxis siguiente:

```
GO TO (n1, n2, n3, ....., nh), i
```

siendo *i* un nombre de variable entera si se trabaja con el FORTRAN básico o bien una expresión entera, real o doble precisión si se trabaja con el FORTRAN completo (*).

n_1, n_2, n_3, ... n_h son números o etiquetas de sentencias ejecutables que se encuentran en el programa.
La coma (,) que precede a la *i* es opcional: puede figurar o no.

---

* Cuando i es real o doble precisión, el compilador considera la parte entera únicamente. Por ejemplo: si fuera 3.2573, tomaría i = 3.

Con esta instrucción existe la posibilidad de bifurcar a diversas sentencias, según sea el valor de la variable *i*. Si dicha variable toma el valor 1, se realizará la bifurcación a la sentencia número n_1; si 2 a la n_2 y si h a la n_h.

Cuando sea $i < 1$ o bien $i > h$, esta instrucción no tiene ningún efecto y se ejecuta la sentencia siguiente en secuencia.

Graficamente, el efecto de esta sentencia podría indicarse de la forma siguiente (fig. 2.3):

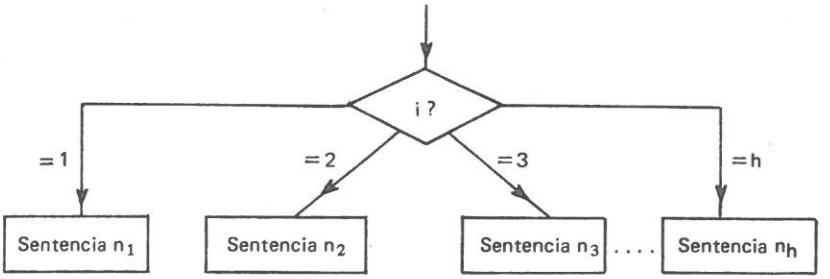


Figura 2.3

El símbolo superior en forma de rombo se emplea en los *diagramas de flujo* para indicar una consulta y una toma de decisión. En este caso: ¿cuál es el valor de la variable *i* ? Según sea este valor, el camino a seguir será uno tomado de entre varias alternativas.

Analicemos el ejemplo siguiente:

```
      GO TO (10,33,1), IKO
        .
        .
        .
1     A = B + 0.2
        .
        .
        .
10    A = B + 3.8 + C
        .
        .
        .
33    ZECO = 3. * (A + B)
```

La primera sentencia es una GO TO calculada. Cuando la variable IKO tome el valor 1, se bifurcará a la sentencia número 10 ejecutándose

```
10  A = B + 3.8 + C
```

Cuando IKO tome el valor 2, se bifurcará a la 33

```
33  ZECO = 3. * (A + B)
```

y cuando tome el valor 3, se bifurcará a la instrucción número 1.

```
1  A = B + 0.2
```

Otro ejemplo. Las sentencias:

```
I = 2
GO TO (23, 40, 57), I
```

harían que el control del programa se transfiriera a la sentencia número o etiqueta 40. Esto indica que el valor de la variable I, debe estar definido previamente a la ejecución de la sentencia GO TO calculada.

Los números de las sentencias encerrados entre paréntesis no necesitan ser distintos; es decir, la sentencia:

```
GO TO (5, 5, 7, 9, 7), J
```

sería válida.

Existe otro GO TO denominado GO TO *asignado* que se utiliza en conjunción con la sentencia ASSING. Ambas sentencias pertenecen al standard del FORTRAN 77, pero se estudiarán más adelante.

2.11. SENTENCIA IF ARITMETICO

En el párrafo 2.9 se hizo aplicación de un conjunto de sentencias FORTRAN para calcular la suma de enteros consecutivos. Pero por limitación de conocimientos, aquellas instrucciones formaban un bucle sin fin. Vamos a introducir una nueva sentencia que vendrá a solucionar este problema.

Otra sentencia que se emplea para realizar bifurcaciones, según ciertas condiciones, es la sentencia IF. Responde a la sintaxis siguiente:

```
IF (e) n1, n2, n3
```

siendo: e una expresión aritmética entera o real, en FORTRAN básico o bien entera, real o doble precisión en FORTRAN completo. n_1, n_2 y n_3 son números o etiquetas de sentencias ejecutables que se encuentran en el mismo programa.

El significado de esta instrucción es: si el valor de la expresión aritmética e es menor, igual o mayor que cero, el control se transfiere, respectivamente, a la sentencia n_1, n_2 o n_3.

Los números externos al paréntesis pueden coincidir alguno de ellos. ellos.

Por ejemplo, en

```
IF (A – B) 3, 22, 45
```

se evaluaría primero la expresión aritmética A–B y dependiendo si su valor es menor, igual o mayor que cero, se bifurcaría a la sentencia número 3 ó a la 22 ó a la 45, respectivamente.

En las ecuaciones algebraicas de segundo grado es obligado calcular el valor del discriminante y compararlo con cero. Esto se conseguiría en FORTRAN así:

```
IF (B * * 2 – 4. * A * C) 100, 200, 300
```

Según que el discriminante sea negativo, nulo o positivo, se bifurcará, respectivamente a las sentencias 100, 200 ó 300.

El flujo representado por una sentencia IF aritmético vendría esquematizado de la forma siguiente (fig. 2.4).

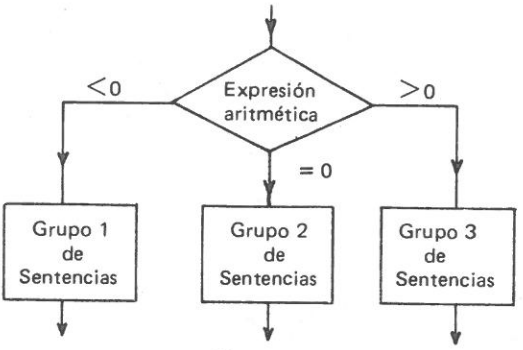


Figura 2.4

Volvamos al ejemplo de 2.9 y supongamos que deseamos calcular la suma

$$1 + 2 + 3 + \ldots\ldots + 20$$

Con la sentencia IF haríamos:

```
    I = 0
    J = 1
1   I = I + J
    J = J + 1
    IF (J – 20) 1, 1, 2
2   STOP
    END
```

Las dos primeras sentencias establecen los valores iniciales que tomarán las variables enteras I y J: cero y uno, respectivamente. Las tres sentencias siguientes se repiten varias veces y por ello se dice que constituyen un bucle. En la primera ejecución del bucle, I toma el valor I = 0 + 1 = 1 y J será J = 1 + 1 = 2. Al llegar a la sentencia IF se calcula la expresión aritmética:

```
J – 20
```

es decir, 2–20 = –18 y por ser negativa, el control del programa bifurcará a la sentencia número 1 donde comienza el bucle de nuevo con I = 1 + 2 = 3, J = 2 + 1 = 3 y J – 20 = 3 – 20 = –17, etc.

Al final del proceso repetitivo, el valor de la suma que deseaba calcularse, se encuentra en la variable entera I y el programa pasará a ejecutar la sentencia número o con etiqueta 2.

Observemos que en el bucle anterior existe un *contador*, el representado por la variable entera J, que se incrementa en una unidad cada vez que se realiza dicho bucle y una instrucción comparativa, la IF, para dar por terminado el proceso o bucle cuando el problema se ha satisfecho. Estas dos situaciones quedan definidas, respectivamente, por las sentencias:

```
J = J + 1
IF (J – 20) 1, 1, 2
```

Programa 2. Volvamos al problema de calcular el área de un círculo. Sean ahora tres círculos de radios respectivos $r_1 = 5{,}9$, $r_2 = 10{,}23$ y

r_3 = 3,29. Haciendo uso de la sentencia IF aritmético, podríamos escribir la siguiente codificación:

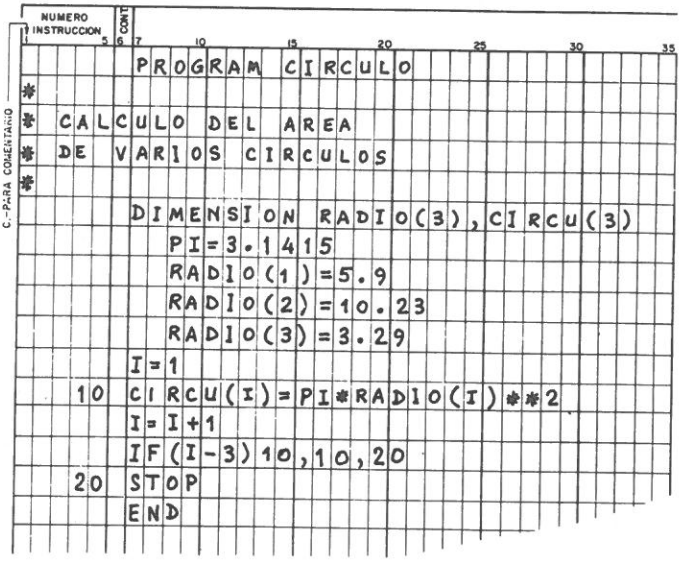


Figura 2.5

La sentencia

```
10 CIRCU (1) = PI * RADIO (I) * * 2
```

es ahora única y sirve para calcular el área de los tres círculos, variando cada vez el subíndice I, desde 1 hasta 3, por medio de la sentencia contador

```
I = I + 1
```

El control del cálculo de cada círculo se efectúa por medio de la instrucción

```
IF (I – 3) 10, 10, 20
```

de modo que cuando sea I>3 finaliza el bucle de cálculo, dándose por terminado el programa.

Programa 3. Supongamos que a un trabajador se le efectúa un descuento de su sueldo mensual, dependiendo de su estado civil, es decir, según sea soltero, casado sin hijos, casado con uno o más hijos, siempre de acuerdo con la tabla siguiente:

Estado	Descuento	Variable de control JJ
Soltero	215,00	= 1
Casado	300,00	= 2
Con 1 hijo	321,75	= 3
Con $\geqslant$ 2 hijos	410,75	= 4

La codificación correspondiente sería:

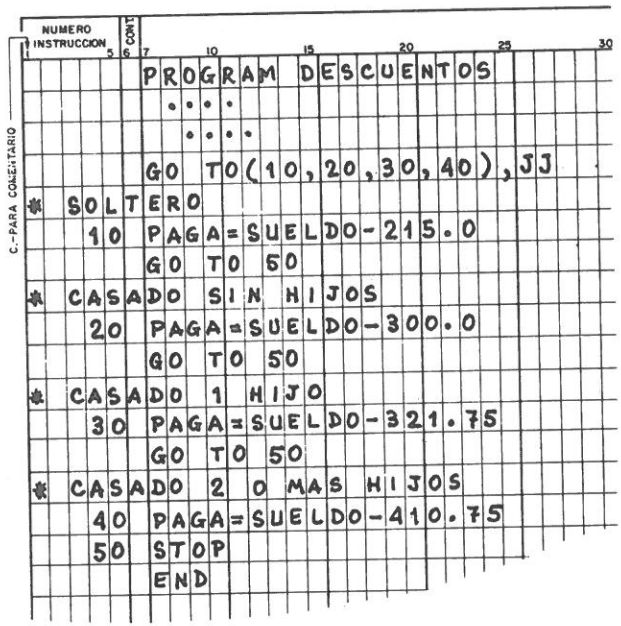


Figura 2.6

En el programa anterior DESCUENTOS, se han escrito una serie de puntos suspensivos que significan: dar valores previos a las variables

SUELDO y JJ la cual sirve para controlar el proceso. Normalmente el valor adjudicado a las variables se hará por medio de sentencias de entrada al programa o, lo que es lo mismo, leyéndolos del exterior. Todo ello se estudiará más adelante. Mediante la sentencia:

```
GO TO (10, 20, 30, 40), JJ
```

se dirigirá el cálculo a la instrucción correspondiente dependiendo de cuál sea el estado civil, siempre de acuerdo con la tabla anterior.

En todos los casos existe una bifurcación incondicional al punto 50, con lo que el programa finaliza.

2.12. SENTENCIA DO

Para ejecutar repetidamente una serie de sentencias en bucle, como ocurría en el ejemplo de la suma de los veinte primeros números enteros, tiene mucha utilidad en el lenguaje FORTRAN el uso de una sentencia especial: la sentencia DO (DO = hacer).

Su sintaxis es:

```
DO n, i = m1, m2, m3
```

siendo: n, una constante entera positiva que indica el número de una sentencia posterior a la DO, que se denomina *sentencia terminal* del bucle DO.

i, es una variable entera, en el FORTRAN básico, o entera, real o doble precisión en el completo. Se denomina *variable del DO*

m_1, m_2, m_3 son constantes o variables enteras, en el FORTRAN básico, o expresiones enteras, reales o doble precisión en el completo. Digamos que indican, respectivamente, un valor inicial, un valor final y un incremento.

La coma (,) entre el número de la sentencia terminal n y la variable del DO i, puede omitirse. La variable m_3 puede también omitirse cuando su valor sea 1.

El conjunto de sentencias que se repiten como bucle y que comienzan en la siguiente a la DO finalizando en la sentencia terminal, recibe el nombre de *rango del DO*. Estos conceptos se indican en el esquema siguiente:

BUCLE DO

SENTENCIA DO n, i = m_1, m_2, m_3
Sentencia 1
Sentencia 2
.
.
.
Sentencia terminal n

RANGO DEL DO

Obsérvese que la sentencia DO no forma parte de su rango.

Como ya hemos hecho anteriormente con otras sentencias, el flujo marcado por una sentencia DO podría indicarse mediante el diagrama siguiente (fig. 2.7):

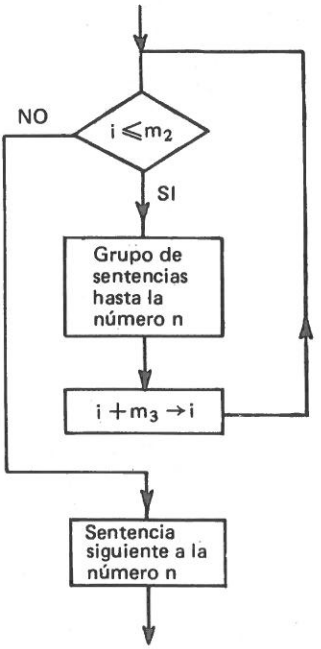


Figura 2.7

En este diagrama, $i + m_3 \rightarrow i$ significa reemplazar i por el valor de $i + m_3$, es decir, sumar m_3 a i, repitiendo a continuación el mismo proceso anterior con ese mismo valor de i.

Veamos detenidamente como opera una sentencia DO. Para ello supongamos el ejemplo:

```
      DO 20 J = 1, 3, 1
      A (J) = B (J) + C (J)
20    D (J) = A (J) + 1.0
```

El rango de este DO está formado por dos sentencias: la segunda de las escritas y la terminal que tiene el número 20. La variable del DO, J, toma el *valor inicial* $m_1 = 1$ y con ese valor se ejecuta el rango, es decir,

```
      A (1) = B (1) + C (1)
20    D (1) = A (1) + 1.0
```

Ahora la variable del DO, J, se *incrementa* en el valor $m_3 = 1$, con lo que pasará a valer 2. Este valor se compara con el *valor final* $m_2 = 3$ y por no ser mayor que él se produce una nueva ejecución de todas las sentencias del rango, es decir:

```
      A (2) = B (2) + C (2)
20    D (2) = A (2) + 1.0
```

De nuevo se incrementa J en 1 (J = 3). Como su valor no supera aún al valor final, vuelven a ejecutarse las instrucciones del rango, así:

```
      A (3) = B (3) + C (3)
20    D (3) = A (3) + 1.0
```

De nuevo J se incrementará en 1 (J = 4) y como ahora esta cantidad supera al valor final, $m_2 = 3$, el rango no se repite más y el programa continúa con la instrucción siguiente a la número 20.

Para dar una regla general, diremos que el número total de veces que se ejecuta el rango de un DO, también denominado *número de iteraciones,* viene dado por la fórmula

$$\text{TOTAL} = \left[\frac{m_2 - m_1}{m_3}\right] + 1$$

donde [.....] indica la parte entera del resultado que se obtenga para la fracción contenida en su interior.

Así, al escribir la sentencia:

```
DO  100  J=2,12,3
```

el número total de veces que se repetiría el rango sería de

$$\text{TOTAL} = \left[\frac{12-2}{3}\right] + 1 = [3{,}33...] + 1 = 3 + 1 = 4$$

es decir, cuatro veces, tomando J los valores sucesivos 2, 5, 8 y 11.

Si el valor TOTAL es cero o negativo, el DO es *inactivo*, es decir, no se ejecuta ninguna vez, (*) lo cual sucede cuando es

$$m_1 > m_2 \quad y \quad m_3 > 0$$

o bien cuando es

$$m_1 < m_2 \; y \; m_3 < 0$$

La variable del DO puede que no figure dentro del rango del mismo, en cuyo caso únicamente sirve para controlar el número de veces que se repiten las sentencias que componen dicho rango. Obsérvese el siguiente ejemplo:

```
      DO  40  I=1,8
      A=A+2.0
40    B=B+3.0
      .
      .
      .
      .
      .
```

En este caso I que es la variable del DO, no figura luego en las dos sentencias que constituyen su rango. Dicho rango se ejecuta sin embargo ocho veces (I = 1, 2, 3, ..., 8). La primera sentencia hace que se incremente en *dos* unidades el valor de la variable A cada vez que se ejecute el bucle; la segunda incrementa en *tres* unidades el valor de la B.

(*) Existen, no obstante, compiladores que aun dándose esas circunstancias, el rango o bucle se ejecuta una vez.

Compruebese que al final los valores de A y B son:

```
A = A + 16.0
B = B + 24.0
```

Recordando el ejemplo propuesto en 2.9 y 2.11 sobre la suma de números enteros consecutivos, si el total de éstos fuera 20, es decir, si hubiera de calcularse

$$S = 1 + 2 + 3 + 4 + \ldots + 20$$

podría realizarse esta suma, escribiendo la codificación:

```
    I = 0
    DO 100 J = 1, 20
1   I = I + J
```

Al final del bucle, el valor de la suma S se encontraría almacenado en la variable o localización I.

2.13. SENTENCIAS DO EN NIDO

Cuando se da la circunstancia de que dentro del rango de una sentencia DO, existen otra u otras sentencias DO con sus rangos respectivos, la reunión de todas ellas se dice que constituyen un conjunto de DO en nido.

Observemos la siguiente codificación y esquema:

```
4    DO   20   K = 1,8,3
1    .
     .
     .
3    DO   30   L = 2,12,2
2    .
     .                      } DO 2º NIVEL  } DO 1er NIVEL
     .
30   A = A + C
31   .
     .
     .
20   DO = 2.0 * E
```

En este ejemplo, el rango de la sentencia DO más externa, que lleva el número 4, comienza en la sentencia número 1 y finaliza en la 20. Todas esas sentencias constituyen un DO de *primer nivel.* Pero dentro de ese rango existe otra sentencia DO, la número 3, cuyo rango comienza en 2 y acaba en la sentencia 30. Estas últimas sentencias constituyen otro DO dentro del primero y por eso decimos que se trata de un DO de *segundo nivel.*

Ambas constituyen un conjunto de sentencias DO en nido.

Veamos cómo se ejecuta el nido del ejemplo. Con el valor de K = 1 comienza a ejecutarse el rango del primer nivel a partir de la sentencia 1. Cuando se llega a la ejecución de la sentencia número 3, comienza un nuevo DO y la realización de un nuevo rango, desde la sentencia 2 a la 30 y con L = 2 hasta L = 12 variando de dos en dos, es decir, este rango se ejecuta.

$$\left[\frac{12-2}{2}\right] + 1 = 5 + 1 = 6 \text{ veces}$$

Una vez satisfecho el rango del segundo nivel, es decir, realizado seis veces, el proceso prosigue en la sentencia 31 que pertenece al rango del DO número 4 y así sucesivamente hasta la sentencia final que lleva el número 20.

En este momento, vuelve a repetirse la ejecución de la sentencia número 1, con K = 4; el DO número 3 se ejecuta de nuevo hasta satisfacerse, es decir, otras seis veces, prosiguiendo luego en la sentencia 31 y sucesivas hasta la 20.

Este proceso se repite

$$\left[\frac{8-1}{3}\right] + 1 = 2 + 1 = 3 \text{ veces}$$

es decir, las que marca la sentencia

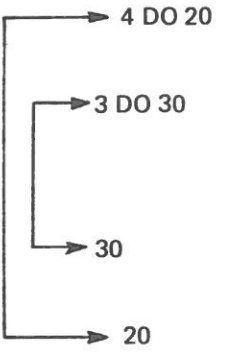
```
4  DO  20  K = 1, 8, 3
```

que gobierna el DO del primer nivel.

Resumiendo, el rango del DO del primer nivel se realiza tres veces, mientras que el rango del DO del segundo se ejecuta dieciocho: seis veces por sí msimo y tres por estar contenido en otro. Además, mientras no finaliza la ejecución de las sentencias de un DO interno no prosigue la ejecución de las sentencias exclusivas del externo.

Obsérvese, y esto es importante, que todas las sentencias del segundo nivel pertenecen, a su vez, al rango del DO del primer nivel.

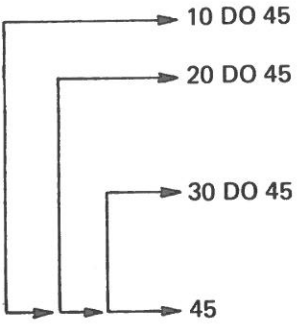
El esquema correspondiente a este ejemplo que se ha analizado, podría ser:



En los ejemplos que siguen se indica el número de veces que se ejecuta cada nivel de rangos. El primer ejemplo es un nido de dos DO y el segundo de tres.

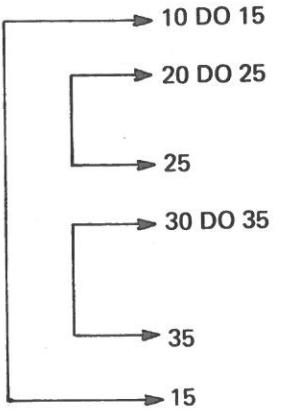
NIDO	Nº del DO	VECES QUE SE EJECUTA
1 DO 20 I = 2,8,5 2 DO 30 J = 1,3	1	$\left[\frac{8-2}{5}\right]$ + 1 = 1 + 1 = 2 veces
	2	$\left[\frac{3-1}{1}\right]$ + 1 = 2 + 1 = 3; total 3 x 2 = 6 veces
1 DO 30 IA = 7,9,3 2 DO 10 JJ = 1,3 3 DO 20 KK = 2,5,2	1	$\left[\frac{9-7}{3}\right]$ + 1 = 0 + 1 = 1 veces
	2	$\left[\frac{3-1}{1}\right]$ + 1 = 2 + 1 = 3; total 3 x 1 = 3 veces
	3	$\left[\frac{5-2}{2}\right]$ + 1 = 1 + 1 = 2; total 2 x 3 x 1 = 6 veces

Utilizando de nuevo el esquema anterior, puede suceder que el nido del DO adquiera la configuración siguiente:



Esta configuración indica tres DO en nido, el 10, el 20 y el 30, pero con la particularidad de que todos ellos acaban en la misma sentencia: en la número 45.

Por fin otra configuración válida sería la representada por:



que indica dos DO, el 20 y el 30 comprendidos en otro que comienza en la sentencia 10 finalizando en la 15. En esta configuración el segundo nivel o nivel más interno está formado por dos DO independientes entre sí.

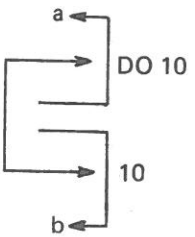
2.14. REGLAS SOBRE EL USO DEL DO

El uso de la sentencia DO en un programa FORTRAN es muy conveniente por la potencia que dicha sentencia posee: pueden realizarse muchos cálculos con poca codificación. Ahora bien; es necesario conocer en profundidad todas las reglas por las que esta sentencia se rige para no incurrir en errores de sintaxis a la hora de compilar o de lógica a la hora de ejecutar el programa al que dicha sentencia pertenece.

Aconsejamos, como mejor regla, estudiar en cada caso la instrucción DO según el manual correspondiente al ordenador donde se vaya a realizar el programa.

Pero antes de exponer las reglas generales del uso del DO vamos a definir dos conceptos.

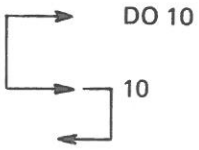
Se dice que en un DO se produce una *salida anormal,* cuando dentro del rango del DO existe una instrucción de bifurcación, como por ejemplo GO TO incondicional, IF aritmético, etc., tal que produce una transferencia de control a otra instrucción que se encuentra fuera del rango del DO. Esquemáticamente se representaría por:



donde *a* o *b* son las flechas que indican una transferencia al exterior.

Se dice que en un DO se produce una *salida normal* cuando se ejecutan las instrucciones pertenecientes al rango del DO hasta que el valor de la variable del mismo supera a m_2 por primera vez, pasando el control de ejecución a la instrucción que sigue inmediatamente a la número *n*. Cuando se produce una salida normal, se dice que el DO se ha satisfecho.

Esquemáticamente sería:



Analicemos ahora las reglas más importantes.

Regla 1. El valor de la variable o constante m_3 de una sentencia DO no puede ser nunca cero.

La siguiente codificación sería inválida

```
      .
      .
J = 4
      .
      .
DO 5  I = 1, K, J – 4
      .
      .
      .
```

Regla 2. La última sentencia del rango de un DO no puede ser ninguna de las siguientes: GO TO incondicional, GO TO asignado, IF aritmético, STOP, END, otra sentencia DO, RETURN. Esta última sentencia será estudiada más adelante.
Para evitar que un rango acabe en estas sentencias puede utilizarse la la sentencia CONTINUE que definiremos próximamente, como instrucción de fin del bucle o iteración.

Regla 3. Cuando se produce una salida normal del rango de un DO, el valor de su índice i, normalmente es igual al primer valor que supera a m_2 siguiendo las reglas de variación de i. Por ejemplo, supongamos la codificación

```
      N = 0
      DO 100  I = 1, 10
      J = I
100   N = N + 1
```

Después de la ejecución de las anteriores instrucciones sería:

I = 11, J = 10 y N = 10

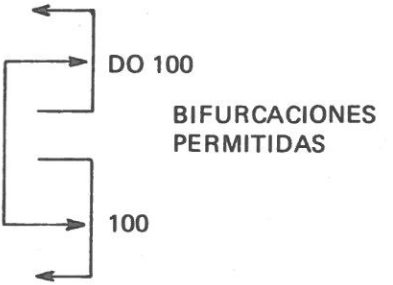
Igualmente, supongamos la codificación:

```
      N = 0
      DO  200    I = 1,10
      J = I
      DO  200    K = 1,5
      L = K
200   N = N + 1
```

Al final de la ejecución será:

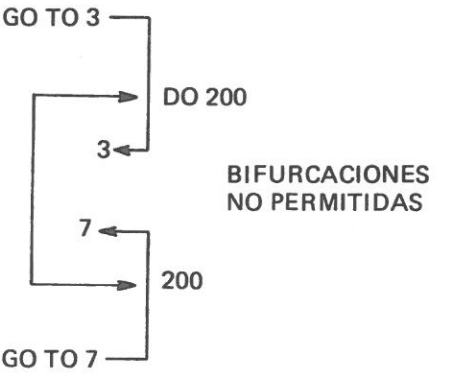
I = 11, J = 10, K = 6, L = 5 y N = 50

Regla 4. Está permitido bifurcar fuera del rango del DO, es decir, efectuar una salida anormal, bien sea hacia adelante o hacia atrás. Esquemáticamente sería:



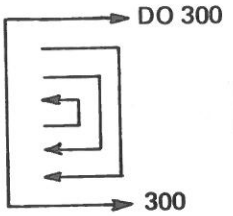
Cuando se produce una salida anormal, el valor de la variable del DO después de la bifurcación es, generalmente, el último valor que haya tomado antes de salir del rango.

Regla 5. No está permitido bifurcar desde fuera hacia dentro del rango de un DO.
El siguiente esquema representa esta regla.



Regla 6. El valor de la variable del DO, i, es usado automáticamente dentro del rango; pero no está permitido redefinirla.

Regla 7. Está permitido todo tipo de bifurcación dentro del propio rango.



BIFURCACIONES PERMITIDAS

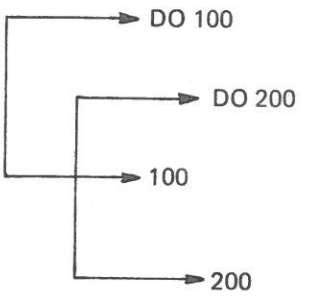
Regla 8. Está permitido cambiar los valores de las variables utilizadas para indicar el valor inicial m_1, el final m_2 o el incremento m_3; pero esos cambios no afectarán al proceso de variación de i establecido ya desde el principio. Por ejemplo estaría permitida la codificación.

```
DO 100  I = N, J, 5
   .
   .
   .
N = N + 2
   .
   .
   .
J = J * 2
   .
   .
   .
100  final del rango
```

donde se modifican los valores de N y de J dentro del rango; pero ello no afectará al valor que tuviera I (variable del DO).

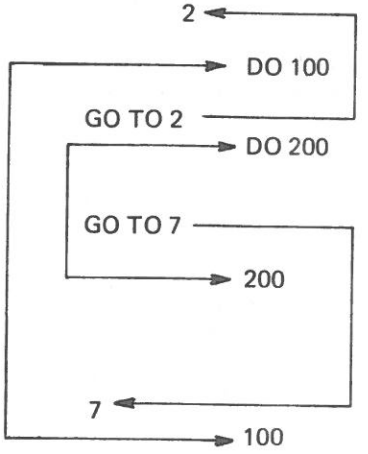
Regla 9. Tratándose de un nido de DO, debe tenerse en cuenta que todas las instrucciones del más interno deben estar contenidas en el rango del externo. Aunque todos ellos pueden compartir la misma sentencia terminal. Véanse los esquemas de las páginas 86 y 87.

Regla 10. No estarían permitidos rangos de DO como los que se indican esquemáticamente a continuación:

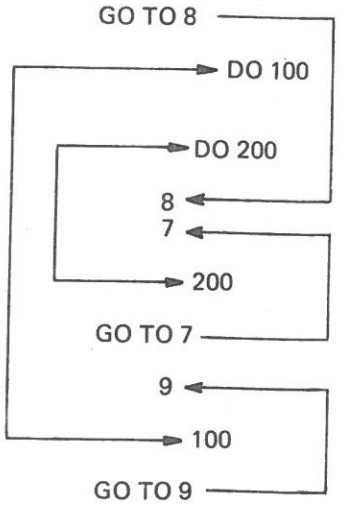


NO PERMITIDO

Regla 11. En los esquemas que se adjuntan quedan expresadas algunas otras reglas permitidas o no.



BIFURCACIONES PERMITIDAS



BIFURCACIONES NO PERMITIDAS

2.15. SENTENCIA CONTINUE

Esta sentencia es ejecutable pero no genera instrucción alguna al ser traducida a código máquina por el compilador. Su uso más corriente es el siguiente. si como se ha dicho anteriormente, Regla 2 del DO, sucediera que la última sentencia de su rango fuera una sentencia prohi-

bida, se emplearía después de ellas la sentencia CONTINUE, cuya sintaxis es simplemente:

```
CONTINUE
```

Véase a continuación una codificación inválida y su correspondiente válida:

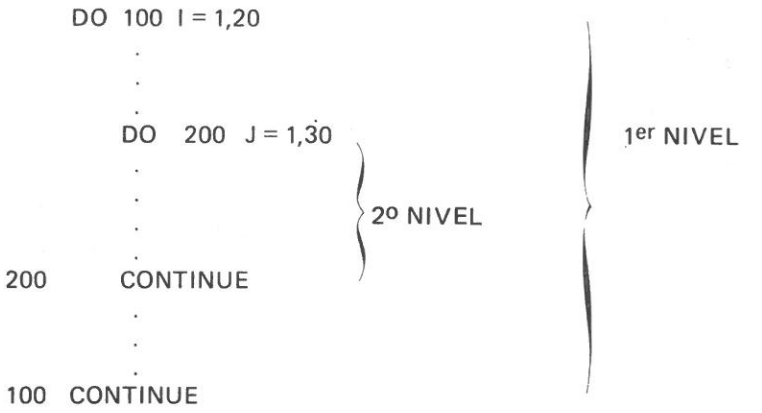
```
DO 3 I = 1, 10                DO 3  I = 1, 10
   .                             .
   .                             .
   .                             .
A (I) = B (I) + 3.0           A (I) = B (I) + 3.0
C (I) = A (I)                 C (I) = A (I)
   .                             .
   .                             .
   .                                GO TO 7
3 GO TO 7                     3  CONTINUE
```

Otra aplicación de esta sentencia. En un programa se prescinde de una sentencia a la que se hacía referencia (sentencia número 7). Basta sustituirla por una CONTINUE con el mismo número. Véase la codificación antes y después de la corrección.

```
   .                             .
   .                             .
   .                             .
GO TO 7                       GO TO 7
   .                             .
   .                             .
   .                             .
7 A = B * 3/2                 7  CONTINUE
```

Como puede observarse la sentencia suprimida y sustituida es la 7 A=B*3/2

Es muy útil utilizar la sentencia CONTINUE como final del rango de un DO, sobre todo cuando existen varios de ellos en nido. De esta forma pueden distinguirse unos de otros con más claridad. Por ejemplo:



2.16. SENTENCIA PAUSE

Para indicar una parada temporal en la ejecución de un programa FORTRAN se utiliza la sentencia PAUSE, cuya sintáxis o forma general es:

```
PAUSE n
```

donde *n* es una serie de cinco dígitos como máximo o bien una *constante carácter.*

Ejemplos de la sentencia PAUSE:

```
PAUSE
PAUSE 2374
PAUSE 'FINAL DE LA RUTINA'
```

La parada temporal que esta sentencia produce, da lugar a que el operador de la máquina o bien el propio usuario del programa sentadc ante un terminal, tomen un tipo determinado de acción como colocar un disco magnético donde se encuentra un conjunto de datos a emplear, colocar papel en una impresora, etc.

Por ello, un uso útil de esta sentencia, sería por ejemplo:

```
      .
      .
      .
PAUSE 'COLOCAR DISCO 1 EN UNIDAD 3'
      .
      .
      .
```

Programa 4. Escribir una codificación con la que se realice la suma de los cien primeros números pares.

```
      PROGRAM PARES
C
C     SUMA DE LOS NUMEROS
C     PARES MENORES QUE 102
C
      I=2
      SUMA=0
   10 SUMA=SUMA+I
      I=I+2
      IF(I-100) 10,10,20
   20 STOP
      END
```

Figura 2.8

Programa 5. Haciendo uso de un bucle **DO**, escribir un programa para calcular la suma de los cuadrados de los números impares comprendidos entre 1 y 10.

```
      PROGRAM CUADRADOS
C
C
      SUMA=0
      DO 10 I=1,10,2
      SUMA=SUMA+I*I
   10 CONTINUE
      STOP
      END
```

Figura 2.9

Obsérvese cómo la variable donde han de acumularse los resultados, en este caso SUMA, inicialmente se le asigna el valor cero.

La sentencia CONTINUE podría haberse evitado, ya que el DO no finaliza en ninguna sentencia prohibida. Unicamente figura en el programa para indicar con claridad el fin de la iteración.

EJERCICIOS RESUELTOS

2.1. Dadas las siguientes expresiones algebráicas, escribir las correspondientes expresiones FORTRAN. Hacer uso de paréntesis para mayor claridad.

a) $a^b + 4gt - 2$

b) $\frac{\pi r^2}{3h}$

c) $\left(\frac{2v^3}{5}\right)^2$

d) $\frac{ab}{x - y^2}$

e) $\frac{2(h - j + 3)}{4v}$

f) $x + \frac{y}{2} + v$

g) $\frac{1}{2} gt^2$

h) $\frac{2}{5} \pi r^2 h$

i) $\sqrt{2gh} = (2gh)^{1/2} = (2gh)^{0,5}$

Respuesta

a) (A ** B) + (4* G * T) – 2
b) (PI * (R**2))/(3 * H) o bien (PI * R ** 2) / (3 * H)
c) ((2 * (V** 3)) / 5) ** 2 o bien ((2 * V ** 3) / 5) ** 2
d) (A * B) / (X – (Y ** 2)) o bien (A * B) / (X – Y ** 2)
e) (2 * (H – J + 3)) / (4 * V)
f) X + (Y/Z) + V o bien X + Y/Z + V
g) (1/2) * G ** (T ** 2) o bien (G * T ** 2)/2 ó 0.5 * G * T ** 2
h) (2 * PI * (R ** 2) * H)/5 o bien 2 * PI * R ** 2 * H/5
i) (2 * G * H) ** 0.5

2.2. Indicar a qué tipo, real, entera o mixta, pertenece cada una de las siguientes expresiones FORTRAN:

a) A
b) 8. * ATO – VELO/3.
c) X ** Y
d) I ** 2 + M (3)
e) RESTO + 4
f) A (I) + 3./B (2 * JAR)
g) X (I) – Y (2)
h) B + 2. – JOB (3)

Respuesta

a) Real " b) Real " c) Real " d) Entera " e) Mixta.
f) Real aunque 2 * JAR es entero (No le da carácter a la expresión por ser un subíndice).
g) Real, aunque I y 2 son subíndices enteros
h) Mixta

2.3. Decir qué tipo de resultado, entero o real, se obtiene una vez efectuados los cálculos indicados por las expresiones siguientes:

a) A + B
b) RAIZ ** (A + 2)
c) A (I) – 6
d) I ** A
e) KM + 2
f) I + 2.
g) A * 3
h) I ** A (I)
i) A/I
j) I/J
k) I * J

Respuesta

a) Real " b) Real " c) Real " d) Real
e) Entero " f) Real " g) Real " h) Real
i) Real " j) Entero con posible truncamiento " k) Entero

2.4. Suponiendo que A = 3.0, B = 2.0, C = –1.0, I = 1 y J = 5, calcular el valor de las expresiones siguientes:

a) A + B
b) I – J
c) A * I
d) A ** I
e) A + B – J
f) B + C + I
g) I ** A
h) A/I

Respuesta

a) 5. ó 5.0
b) –4
c) 3. ó 3.0
d) 3. ó 3.0
e) 0. ó 0.0
f) 2. ó 2.0
g) 1. ó 1.0
h) 3. ó 3.0

2.5. Calcular el valor de las expresiones siguientes, aplicando en cada una de ellas las prioridades correspondientes:

a) 5 + 4.8 * 3
b) 2. * 3 ** 2
c) 6 / (4 * 2.)
d) 6/ (4 * 2)
e) – 7/3
f) 6.8 + 15/7
g) (7/3) / (7/5)

Respuesta

a) 5 + 4.8 * 3 = 5 + 14.4 = 19.4
b) 2. * 3 ** 2 = 2. * 9 = 18.0
c) 6/ (4 * 2.) = 6/8. = 0.75
d) 6/ (4 * 2) = 6/8 = 0 (truncamiento)
e) −7/3 = 2 (truncamiento)
f) 6.8 + 15/7 = 6.8 + 2 = 8.8 (truncamiento al realizar 15/7)
g) (7/3) / (7/5) = 2/1 = 2 (truncamiento en 7/3)

2.6. En cada uno de los ejercicios que siguen se dan, emparejadas, una expresión algebráica y su correspondiente FORTRAN. Decir cuál o cuáles son los errores que contienen estas últimas.

a) $\left(\frac{a}{b}\right)^{c+3}$ ⇒ (A/B) ** C + 3

c) $\frac{a \cdot b}{c\,(d+c)}$ ⇒ AB/ C * (D + C)

b) $\left(\frac{a+b}{b}\right)^{2}$ ⇒ (A+B/B) ** 2

d) $\frac{x^{n+2}}{y^{n-1}}$ ⇒ (X**N +2)/ Y ** N − 1

Respuesta

Las expresiones FORTRAN correctas serían:

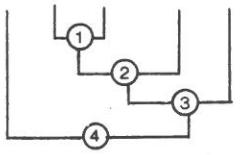
a) (A/B)*(C + 3)
b) ((A + B) / B) ** 2
c) (A * B)/ (C * (D + C))
d) X ** (N + 2) / Y ** (N − 1)

2.7. Indicar el orden en que se realizan las operaciones en las expresiones FORTRAN siguientes:

a) A + B * 2. / C (I) * 3.
b) A ** B ** C/4. * L (I) + D
c) A + (B * C) / D
d) (A + B * C) / (3. + D) + 2.
e) A + ((B + C) * 3.) + VELO ** 2

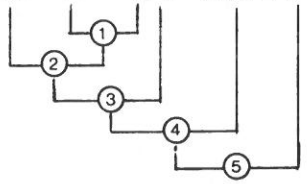
Respuesta

a) A + B * 2. / C (I) * 3.



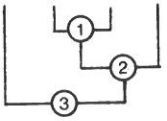
*
/
*
+

b) A ** B ** C/4. * L (I) + D



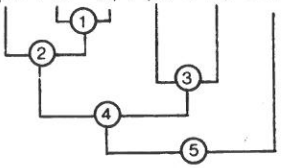
**
**
/
*
+

c) A + (B * C) / D



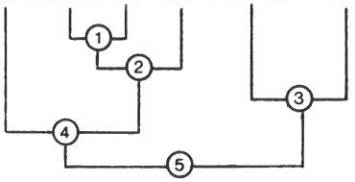
*
/
+

d) (A + B * C) / (3. + D) + 2.



* \
\+ \ 1er Paréntesis
\+ 2º Paréntesis
/
+

e) A + ((B + C) * 3.) + VELO ** 2



\+ 1er paréntesis
* 2º paréntesis
**
+
+

2.8. Se da la siguiente lista de valores numéricos:

1028, −325, 487, 518, −1535

Suponiendo que dicho conjunto se denomina por la variable TABLA, decir, qué valores corresponden a:

a) TABLA (5)
b) TABLA (2)
c) TABLA (7)
d) TABLA (4)
e) TABLA (8)

Respuesta

a) −1535
b) −325
c) No existe. El conjunto únicamente tiene cinco elementos
d) 518
e) No existe

2.9. Escribir la sentencia DIMENSION correspondiente para declarar o definir el conjunto anterior TABLA

Respuesta

```
DIMENSION TABLA (1:5)
```

o bien

```
DIMENSION TABLA (5)
```

2.10. Indicar el tipo de conjunto, entero o real, y el número de dimensiones del mismo al emplear las siguientes variables subindicadas:

a) INTER (2,8)
b) NUMERA (10, 3, 1)
c) POLAR (8)
d) CIMA (4, 5, 7, 3)
e) VELOC (5, 7, 1, 2)

Respuesta

a) Entero de 2 dimensiones
b) Entero de 3 dimensiones
c) Real de 1 dimensión
d) Real de 4 dimensiones
e) Real de 4 dimensiones

2.11. Dadas las siguientes sentencias, contestar a las preguntas que se indican:

a) DIMENSION AA (1:4)
b) DIMENSION B (1:8, 2, 4)
c) DIMENSION C (2,7)
d) DIMENSION DD (8, 10)
e) DIMENSION X (1:3, 1:4, 1:3)
f) DIMENSION VEL (1:5, 2, 8, 10)

Nombre del conjunto que se declara; dimensiones que tiene el conjunto; número máximo de elementos del mismo.

Respuesta

a) AA; 1; 4
b) B; 3; 64 (= 8 x 2 x 4)
c) C; 2; 14 (= 2 x 7)
d) DD; 2; 80 (= 8 x 10)
e) X; 3; 36 (= 3 x 4 x 3)
f) VEL; 4; 800 (= 5 x 2 x 8 x 10)

2.12. Se da la sentencia DIMENSION A(4,2). Decir qué lugares, dentro del conjunto, ocuparán los elementos siguientes:

a) A (3,1)
b) A (4,2)
c) A (3,2)
d) A (1,2)
e) A (2,3)

Respuesta

a) El tercer lugar: 3 + 4 (1–1) = 3
b) El octavo lugar: 4 + 4 (2–1) = 8
c) El séptimo lugar: 3 + 4 (2–1) = 7
d) El quinto lugar: 1 + 4 (2–1) = 5
e) Este elemento no pertenece al conjunto, ya que el segundo subíndice, 3, es superior al valor máximo 2.

2.13. Se da la sentencia DIMENSION AA (1:2, 1:3, 1:2). Indicar qué lugares, dentro del conjunto, ocuparán los elementos siguientes:

a) A (1, 2, 1)
b) A (3, 1, 1)
c) A (1, 2, 2)
d) A (1, 2, 0)
e) A (2, 3, 2)
f) A (2, 1, 2)

Respuesta

a) El tercer lugar: 1 + 2 (2–1) + 2 x 3 (1–1) = 3
b) Este elemento no pertenece al conjunto, ya que el primer subíndice, 3, es superior al valor máximo 2.
c) El noveno lugar: 1 + 2 (2–1) + 2 x 3 (2–1) = 9
d) Este elemento no existe: tiene un subíndice nulo.
e) El lugar doce: 2 + 2 (3–1) + 2 x 3 (2–1) = 12
f) El octavo lugar: 2 + 2 (1–1) + 2 x 3 (2–1) = 8

2.14. Suponiendo que las variables que se indican tienen los valores siguientes:

I = 3 " J = 2 " A = 1.0 " B = 2.5 " C = 10.0

encontrar el valor que toma la variable del primer miembro en las sentencias aritméticas que siguen:

a) ALFA = I
b) VEL (3) = I – J
c) C = 3
d) D = B/C
e) JOTA = B/C
f) N = (C + 2.0 * B) ** I
g) A = 3 ** J + 2 * (I ** J)

Respuesta

a) 3.0
b) 1.0
c) 3.0
d) 0.25
e) 0 (truncamiento)
f) 3375
g) 27.0

2.15. Considérense las siguientes sentencias de asignación como pertenecientes a un mismo programa y cuya ejecución se realiza de forma consecutiva. Sabiendo que como valores iniciales es A = 5.0, B = 2.0 y C = 1.0, deducir los valores de A, B y C después de la ejecución de cada sentencia de asignación.

```
A = A + B
C = C + 3.
A = 2 * B + C
C = C + A * B
A = C/B + 1
B = C/(–5)
```

Respuesta

A	B	C
7.0	2.0	1.0
7.0	2.0	4.0
8.0	2.0	4.0
8.0	2.0	20.0
11.0	2.0	20.0
11.0	-4.0	20.0

2.16. Suponiendo que inicialmente es J = 2, deducir el valor de I siguiendo el programa siguiente:

```
J = J + 1
B = J
B = B + 2.3
I = B
```

Respuesta

```
I = 5
```

2.17. Examinando las sentencias de asignación siguientes, indicar los valores que van tomando las variables sucesivas del lado izquierdo.

```
A = 2
B = 1
C = A + B
D = C * 2 – 3.
V (2) = (D + A) / 2
```

Respuesta

```
A = 2.0
B = 1.0
C = 3.0
D = 3.0
V (2) = 2.5
```

2.18. Se tiene la variable A = 3.58. Escribir un programa que separe las partes entera (3) y fraccionaria (0,58) de ese valor.

Respuesta

```
A = 3.58
I = A
B = A – I
```

ya que será: A = 3.58 " I = 3 (truncamiento) y B = 0.58'

2.19. Se da la codificación siguiente:

```
   X = 1.0
   SUMA = 0.0
6  SUMA = SUMA + X
   X = X + 1.
   GO TO 6
```

Las instrucciones número 6 y siguientes constituyen un bucle. Calcular los seis primeros valores de SUMA y de X.

Respuesta

1) SUMA = 0.0 ,, X = 1.0
2) SUMA = 1.0 ,, X = 2.0
3) SUMA = 3.0 ,, X = 3.0
4) SUMA = 6.0 ,, X = 4.0
5) SUMA = 10.0 ,, X = 5.0
6) SUMA = 15.0 ,, X = 6.0

2.20. Se dan los siguientes GO TO calculados. Deducir la instrucción que se ejecutará dependiendo del valor de las variables que se indican

a) GO TO (1, 5, 7, 9), JJ para JJ = 3
b) GO TO (5, 5, 7, 4, 9, 9), KK para KK = 5
c) GO TO (1, 5, 3), L para L = 4
d) GO TO (2, 2, 2, 5, 1, 100), 4

Respuesta

a) La número 7
b) La número 9
c) Error, ya que sólo existen tres posibilidades 1, 5 y 3 y L vale 4
d) Incorrecto : i debe ser una variable y en este caso es la constante 4.

2.21. Dado el programa siguiente, indicar el contenido de las variables X y SUMA después de finalizada su ejecución.

```
      X = 1.0
      SUMA = 0.0
10    SUMA = SUMA + X * X
      X = X + 3.0
      IF (X - 15.0) 10, 10, 20
20    STOP
      END
```

Respuesta

X	SUMA
1.0	0.0
4.0	1.0
7.0	17.0
10.0	66.0
13.0	166.0
16.0	335.0

Valores finales: X = 16.0
SUMA = 335.0

2.22. Idem que el ejercicio anterior indicando los valores finales de I, X y SUMA.

```
      I = 1
      X = 1.0
      SUMA = 1.0
10    SUMA = SUMA * X
      X = X + 1.0
      I = I + 1
      GO TO (10, 10, 10, 20), I
20    STOP
      END
```

Respuesta

I	X	SUMA
1	1.0	1.0
2	2.0	1.0
3	3.0	2.0
4	4.0	6.0

Valores finales: I = 4
X = 4.0
SUMA = 6.0

2.23. Dadas las siguientes sentencias DO, deducir el número de veces que se ejecutan sus respectivos rangos.

a) DO 33 I = 1, 5, 2
b) DO 42 J = 4, 8, 5
c) DO 123 KILO = 1,5
d) DO 3 LOC = 3,3,2
e) DO 100 J = 2, 12, 3
f) DO 10 I = 1, 100, 0
g) DO 20 I = 1, 10, – 1
h) DO 50 L = –1, –100, –1

Respuesta

a) Tres veces
b) Una vez
c) Cinco veces
d) Una vez
e) Cuatro veces
f) No está permitido $m_3 = 0$
g) Este bucle se ignora (TOTAL = –8)
h) Cien veces

2.24. Idem para las sentencias siguientes:

a) DO 100 I = – 0.5, 1.5, 0.1
b) DO 50 J = 30, –30, –2
d) DO 200 K = 0.5, 5.7, 2

Respuesta

a) 21 veces
b) 31 veces
d) 3 veces

2.25. Calcular el número de veces que se ejecuta el rango correspondiente al nivel más interno de los DO en nido siguientes:

a)
```
DO 20 L = 1, 2, 3
DO 1  I = 4,2
```

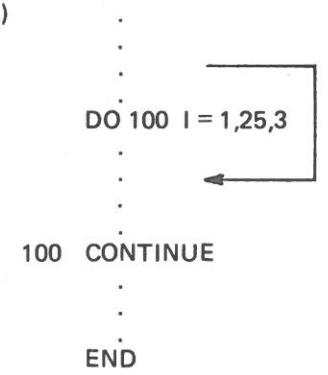
b)
```
DO 10  I = 1, 10, 3
DO 10  J = 2, 12, 2
```

Respuesta

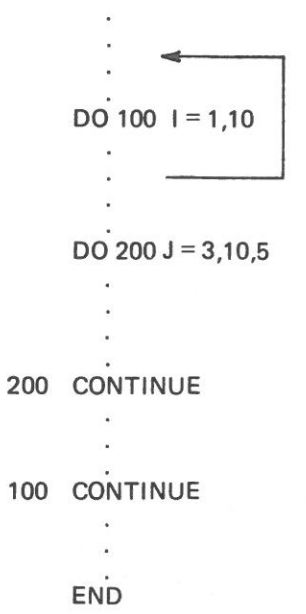
a) Una vez (= 1 x 1)
b) 24 veces (= 4 x 6)

2.26. Dada la siguiente codificación de DO y nido de DO, deducir los errores que puedan existir en las bifurcaciones que se señalan

a)



b)



c)

```
      DO 10  I = 1,10
       .
       .
       .
      DO 100 J = 1,20
       .
       .
       .
       .
       .
10    CONTINUE
       .
       .
       .
100   CONTINUE
       .
       .
      END
```

d)

```
       .
       .
       .
       .
      DO   1 I = 2,10,3
       .
       .
       .
       .
      DO   2 J = 3,7
       .
       .
       .
       .
2     CONTINUE
       .
       .
       .
1     CONTINUE
```

Respuesta

a) Incorrecto. No puede hacerse una transferencia al interior de un rango desde el exterior.
b) Es una bifurcación correcta.
c) Incorrecto. Las sentencias del DO interior no están contenidas todas ellas en el exterior. La bifurcación sería correcta.
d) Incorrecto. No se puede bifurcar a un rango interior desde otro exterior.

3

Lectura y escritura de la información. Ficheros

Nada se dice que no haya sido dicho antes

TERENCIO *(190-159 A.C.)*

3.1. ENTRADAS Y SALIDAS

En el capítulo precedente, se han examinado algunos ejemplos codificados, como la suma de números enteros o el cálculo de la superficie de varios círculos. Estos ejemplos tenían una limitación común: el total de números enteros era fijo; el total de círculos estaba determinado dentro del propio programa. Además, no existía una comunicación entre el programa-ordenador y la persona que había diseñado tal codificación; es decir, no existía una instrucción mediante la cual los resultados, suma total o áreas, fueran comunicados o "dichos" al programador.

Volviendo al ejemplo de 2.11, página 77, un programa mas general que el escrito en aquella ocasión sería:

```
      LEER ENTERO N
      I = 0
      J = 1
1     I = I + J
      J = J + 1
      IF (J – N) 1, 1, 2
2     ESCRIBIR EL VALOR DE LA SUMA I
      STOP
      END
```

Este programa contiene dos instrucciones nuevas: una instrucción de entrada al programa, facilitándole el dato N, variable, mediante la cual se le instruye sobre el número entero más alto a sumar; una instrucción de salida, mediante la cual el programa comunica al usuario cuál es el valor de la suma total I. La instrucción IF compara ahora el valor del contador J con el dato variable final N.

Las instrucciones LEER y ESCRIBIR, tal como figuran en el programa anterior no son aún instrucciones FORTRAN; pero este lenguaje, como veremos más adelante, posee instrucciones equivalentes para lograr esos objetivos.

Volviendo a la figura 1.1, existen varias formas de comunicar al ordenador los datos con los que se desea trabajar: perforarlos en fichas y leer éstas a través de una lectora (*); introducirlos directamente a traves de un terminal pantalla-teclado; tenerlos previamente grabados en un disco o cinta magnéticos e introducirlos después en la Unidad Central de Proceso. Todos estos dispositivos son ejemplos de *Unidades de Entrada.*

Para recibir los resultados de un cálculo, obtenidos por un ordenador de acuerdo con un programa escrito en FORTRAN, pueden utilizarse entre otros los siguientes medios: una impresora; un terminal pantalla-teclado donde pueden visualizarse; grabarlos en una cinta o disco magnéticos y luego interpretarlos. Estos dispositivos son ejemplos de *Unidades de Salida.*

Por consiguiente, existen unos medios denominados dispositivos de Entrada-Salida o abreviadamente, de E/S, que sirven para ingresar la información dentro de la memoria del ordenador o para que éste transmita al exterior los resultados obtenidos.

En correspondencia con estos medios, el lenguaje FORTRAN, al igual que otros lenguajes, dispone de sentencias de E/S.

Toda *sentencia de entrada* facilita la forma de transferir los datos desde un medio externo, fichas, terminales o desde una memoria auxiliar o secundaria, disco, cinta, a la memoria principal. Este proceso se denomina *lectura.*

Toda *sentencia de salida* facilita la forma de transferir los resultados (datos también) desde la memoria principal a una memoria auxiliar o secundaria o a un medio externo, impresora, terminales. Este proceso se denomina *escritura.*

Además de estas sentencias de E/S, el lenguaje FORTRAN dispone de otras, denominadas *sentencias auxiliares de E/S*, que sirven para manipular los medios externos: mover el papel de una impresora, hacer retroceder una cinta magnética o para describir las propiedades que deben poseer esos medios externos.

El total de sentencias de E/S y auxiliares es de nueve:

(*) Sistema prácticamente ya en desuso pero que utilizaremos con frecuencia en nuestros ejemplos por describir muchas nociones con mucha simplicidad.

a) *Sentencias de E/S*

READ, para leer datos desde soportes externos hacia la memoria principal o desde memorias auxiliares a la principal.

WRITE, para escribir o transferir datos desde la memoria principal hacia soportes externos o memorias auxiliares.

PRINT, posee parecida misión que la anterior. Esta sentencia no se encuentra disponible en el FORTRAN 77 básico.

El lenguaje FORTRAN 77 es un lenguaje que básicamente trabaja con registros en cuanto a entradas y salidas se refiere. Esto significa que una sentencia READ da lugar a la lectura de un registro lógico y las sentencias WRITE o PRINT causan la escritura o impresión de un registro lógico. El concepto de registro será estudiado a continuación.

b) *Sentencias auxiliares de E/S*

OPEN, CLOSE, INQUIRE, BACKSPACE, ENDFILE y REWIND.

Todas ellas permiten: manejar el soporte externo o las memorias auxiliares; conocer las propiedades de la conexión a esos soportes o memorias; describir dichas propiedades.

El FORTRAN 77 básico no posee las instrucciones CLOSE e INQUIRE y con respecto a las otras cuatro auxiliares, hemos de decir que existen algunas diferencias en cuanto a su sintaxis y semántica en relación con las mismas pertenecientes al FORTRAN 77 completo.

3.2. CAMPO, REGISTRO, FICHERO

Para llegar a entender correctamente las instrucciones de E/S, es necesario que primero definamos un conjunto de nociones básicas de Proceso de Datos, íntimamente ligadas a todas las operaciones e instrucciones de entrada y salida.

Al hablar de datos de un problema o de resultados obtenidos a través de un programa, manejamos con frecuencia conceptos como: campo, registro y fichero.

a) Campo

Supongamos la estructura de datos mostrada en la figura 3.1.

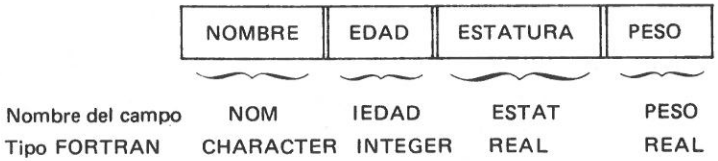


Figura 3.1

En ella están contenidos: el nombre de una persona y varias características correspondientes a la misma, como edad, estatura y peso. Cada uno de estos cuatro datos recibe el nombre de *campo*. Así, tendremos el campo nombre, que en FORTRAN puede denominarse por NOM definiéndolo como variable CHARACTER; el campo edad, IEDAD, como variable entera y los campos estatura y peso, ESTAT y PESO, como variables reales. En total cuatro campos.

Campo es, por consiguiente, la unidad de información más pequeña e indivisible que puede manejarse en FORTRAN.

b) Registro

Un *registro* es un elemento de datos compuesto por un conjunto de campos. Ya puede comprenderse que el registro más simple es el formado por un solo campo.

En el ejemplo anterior teníamos un registro correspondiente a una determinada persona y compuesto por cuatro campos. Los campos pueden ser enteros, reales, doble precisión o conjunto de caracteres alfanuméricos.

Otro ejemplo de registro orientado a una aplicación de nómina en una compañía, viene esquematizado en la fig. 3.2.

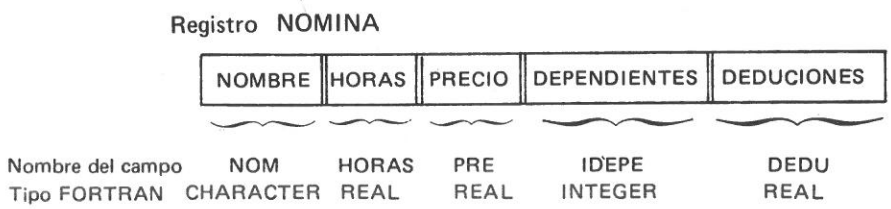


Figura 3.2

Resumiendo: la forma de describir un registro en FORTRAN se hace indicando el nombre de cada uno de sus campos y el tipo que les corresponde respectivamente. No existe, por consiguiente, una manera para denominar formalmente en FORTRAN un registro con una sola variable o un sólo nombre. Así pues, para denominar el registro NOMINA del ejemplo anterior, ha de enunciarse todo su contenido.

No obstante y para que la codificación FORTRAN fuera más clara, podríamos declarar los campos que forman el registro NOMINA empleando las sentencias de especificación siguientes al principio del programa:

```
C     DEFINICION REGISTRO NOMINA
      CHARACTER NOM * 25
      REAL HORAS, PRE
      INTEGER IDEPE
      REAL DEDU
C     FIN DEFINICION REGISTRO
```

Las sentencias REAL e INTEGER serán estudiadas en el Capítulo 6.

Imaginemos que ese registro se ha obtenido como resultado de un programa a través de una impresora del ordenador. Supongamos que fuera

```
JUAN LOPEZ    125.23    200.00    3    3280.75
```

ocupando en total 45 posiciones, entre blancos y caracteres, de una línea de impresora. Normalmente, en una de estas líneas caben hasta 120 ó 130 carácteres. Pues bien; cada línea de impresora se denomina también *registro físico;* mientras que el conjunto de caracteres que constituyen el registro NOMINA de la nómina, recibe el nombre de *registro lógico.*

El registro lógico del primer ejemplo estaba formado por todos los caracteres alfanuméricos correspondientes a nombre, edad, estatura y peso.

Una línea de terminal pantalla-teclado es un registro físico; la información contenida en ella, un registro lógico. Ambos pueden ser de la misma longitud.

Una ficha de ochenta columnas, es otro ejemplo de registro físico; la información contenida (perforada) en ella, un registro lógico. Es posible que no ocupe las ochenta columnas.

Por consiguiente, un registro lógico no siempre coincide con la longitud de la entidad física donde está contenido: puede ser más corto o más largo.

Cuando en FORTRAN 77 se habla de registros se refiere siempre a registros lógicos en oposición al concepto de registro físico.

Aunque ello será descrito más adelante, existe otra clasificación de registros al ser tratados por las instrucciones de E/S del FORTRAN:

Registros con formato
Registros sin formato
Registros de fin de fichero

c) Fichero

Un solo registro lógico no constituye la nómina de una compañía, la cual estará formada por un conjunto, quizás de cientos de estos registros lógicos. Cada registro tendrá la misma estructura de campos, pero el contenido de cada uno de ellos será diferente según el trabajador de que se trate.

Se denomina *fichero* o *fichero de datos* a un conjunto de registros de la misma estructura (Fig. 3.3). Como se muestra en la figura, un fichero esencialmente es un conjunto unidimensional o lineal de registros lógicos. Eso es lo que ocurre en el fichero NOMINA.

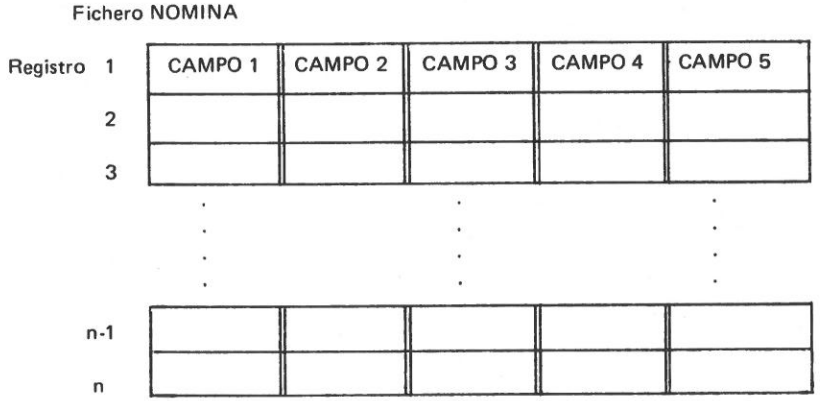


Figura 3.3

Cada registro de un fichero, en general, posee cientos de caracteres y por consiguiente ocupará cientos de posiciones de memoria. Tanto

más ocuparán los ficheros formados por cientos o miles de estos registros. Por ello y para economizar memoria principal del ordenador, los ficheros se almacenan en memorias auxiliares externas a la Unidad Central de Proceso, como cintas y discos magnéticos, siendo transferidos (leídos) a la memoria principal únicamente para su proceso y devueltos (escritos) a la memoria periférica (Fig. 3.4) una vez procesados, es decir, una vez realizados con ellos algún tipo de cálculo. y ello, normalmente, registro a registro.

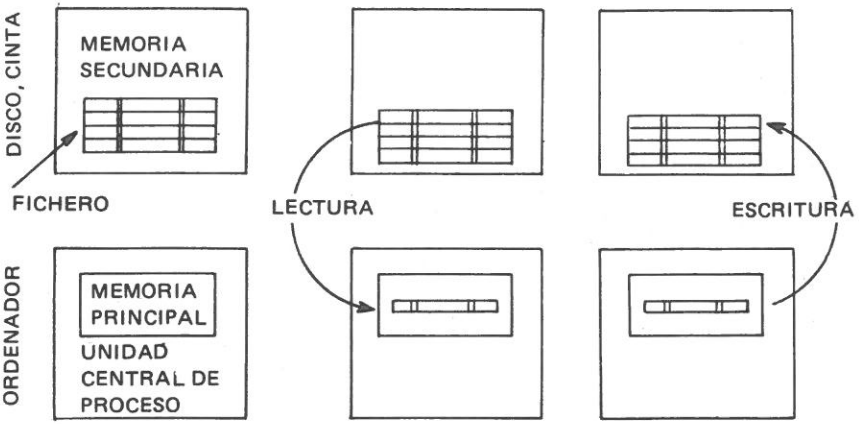


Figura 3.4

Como ya hemos dicho, el lenguaje FORTRAN dispone de un conjunto de sentencias de E/S para llevar a cabo las operaciones descritas de lectura o escritura.

3.3. ACCESO A UN FICHERO

Existen dos formas básicas de obtener o acceder a un determinado registro de un fichero contenido en una memoria auxiliar o secundaria: acceso secuencial o acceso directo. Vamos a describir cada uno de ellos, pero recordando siempre que acceder a un registro es sinónimo de: buscar dicho registro en un fichero y llevarlo a memoria principal para su proceso.

El *acceso* es *secuencial,* cuando, para llegar a un registro dentro de un fichero, es necesario recorrer en secuencia todo el fichero desde el principio hasta encontrar el registro deseado, el cual se reconocerá por

contener un dato particular denominado *clave* que le distingue de los demás y es la que desea procesarse. Por ejemplo, en el caso de la nómina, la clave de cada registro, podría ser el número de empleado y el registro a procesar podría ser el de clave 1285.

Un fichero, cuyos registros están dispuestos según la secuencia ascendente o descendente de su campo *clave*, se dice que es un *fichero sècuencial.*

Los ficheros secuenciales pueden utilizarse sobre cualquier medio de almacenamiento: un conjunto de fichas es un fichero secuencial; lo mismo ocurre con un disco magnético o un diskette. Pero el medio más importante es la cinta magnética, donde el único modo de acceder y procesar sus registros es siguiendo el orden físico en el que están grabados en ella. Por ello se dice que los ficheros sobre cinta magnética tienen que ser ficheros cuyo método de acceso es secuencial (En Inglés *Sequential Access Method* o abreviadamente, ficheros SAM).

El método secuencial es conveniente cuando se trata de ficheros con pocos registros o de ficheros con muchos registros y que vayan a procesarse todos o la mayoría de ellos.

Cuando el fichero consta de muchos registros o bien se procesan únicamente algunos de ellos, es más práctico y conveniente utilizar otro método de acceso denominado *acceso directo* o *acceso al azar*. En este método y una vez conocida la clave del registro deseado, se accede al mismo de forma directa e inmediata sin pasar por todos los que le preceden. Esto ahorra mucho tiempo de acceso.

Un fichero de acceso directo tiene necesariamente que estar almacenado en un disco magnético, diskette o tambor magnético. El método de acceso directo, se denomina en Inglés DAM, de *Direct Access Method.*

En la figura 3.5 se esquematiza la organización física y lógica de un fichero con acceso directo.

Obsérvese como en la organización lógica cada registro posee asociado con él un número en secuencia, comenzando en 1. En la organización física los registros se graban concéntricamente pero no necesariamente en orden consecutivo, sino donde exista sitio libre. No obstante, los discos pueden también utilizarse para grabar en ellos la información, los registros, de forma secuencial.

Más adelante se estudiarán las sentencias de E/S FORTRAN para cada uno de estos tipos de acceso, ya que el lenguaje FORTRAN 77 soporta únicamente estas dos clases de organización, es decir, secuencial y directa.

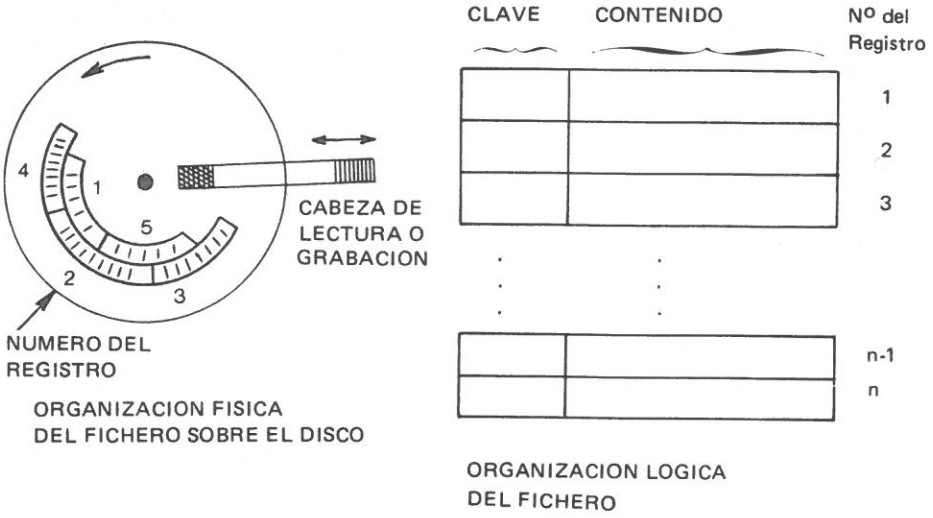


Figura 3.5

3.4. OPERACIONES SOBRE UN FICHERO

Antes de pasar a estudiar las sentencias de E/S, debemos todavía definir algunos otros conceptos que más tarde vamos a emplear en ellas.

Para que los programas escritos en FORTRAN puedan acceder a los ficheros que se vayan a procesar, es necesario hacerlo a través de unidades. Una *unidad* es como un canal mediante el cual el programa tiene acceso a ficheros localizados en memorias o soportes externos: lectoras de fichas, impresoras, diskettes, discos, cintas magnéticas, etc. Cada unidad se identifica empleando un número entero. Por ejemplo: un terminal pantalla-teclado podría ser la unidad 1; una impresora, la 2; una unidad de disco magnético, la 3 y así sucesivamente.

Esto indica, por ejemplo, que para trabajar con un registro de un fichero contenido sobre disco, debería informarse al sistema de cálculo que la unidad a la que debe accederse, es la número 3.

Los números dados anteriormente a las unidades, son ejemplos y debe consultarse en cada caso el manual correspondiente al ordenador utilizado, para saber cuál es la numeración concreta a ese sistema.

El primer registro de un fichero se denomina *registro etiqueta* y contiene entre otras cosas: nombre del fichero, fecha de creación,

fecha a partir de la cual ya no será necesario conservarlo, etc. y todo ello por razones de seguridad.

Si el fichero es secuencial, al final del mismo debe encontrarse un registro especial, que ya hemos mencionado anteriormente, denominado *registro de fin de fichero* (En inglés, END OF FILE).

Cuando desde un programa se utiliza un fichero, la primera operación a realizar es la apertura del mismo. *Abrir* un fichero (en inglés, OPEN) es establecer una conexión con el mismo desde la memoria principal. Normalmente esta conexión se lleva a cabo antes de que el programa comience empleando *comandos del sistema,* independientes del lenguaje FORTRAN.

Pero cuando se trata de FORTRAN 77, este lenguaje posee una instrucción especial, OPEN, que estudiaremos en detalle, con la cual la conexión puede establecerse durante el proceso del programa y antes que el fichero comience a usarse. Abrir un fichero significa pues abrir una conexión entre un determinado fichero cuyo nombre nos es conocido y la unidad física o dispositivo sobre el que reside, de modo que toda referencia posterior a la unidad en cuestión sea considerada como una referencia al propio fichero.

Al abrir un fichero ha de indicarse al ordenador, entre otras cosas: número de la unidad, nombre del fichero, tipo de acceso (secuencial o directo), si se trata de una lectura o de una escritura, etc.

Una vez finalizado el empleo de un fichero, este debe cerrarse. *Cerrar* un fichero (en inglés, CLOSE) es romper la conexión con el mismo. Para ello FORTRAN 77 emplea la instrucción CLOSE. A partir del cierre, el fichero queda inaccesible a menos que vuelva a emplearse otra instrucción OPEN para abrirlo de nuevo.

3.5. SENTENCIAS READ BASICA Y PRINT

Como hemos dicho al principio de este capítulo, para que un programa pueda estar completo, es necesario, en la mayoría de los casos, tener la posibilidad de traer los datos dentro de la memoria del ordenador o de escribir los resultados a través de un medio externo.

Para introducir datos se emplea la instrucción READ y para escribir resultados la PRINT. Esta última únicamente existe en el FORTRAN 77 completo.

La sentencia READ más sencilla o básica utilizable en ambos FORTRAN, tiene la sintaxis o forma general siguiente:

```
READ f, lista
```

siendo f, un asterisco *
o bien una constante o variable entera (incluyendo en esta categoría el nombre de un conjunto), la etiqueta de una sentencia de formato FORMAT, existente en el mismo programa, o bien una constante CHARACTER.

lista es el conjunto de nombres de variables que van a ser leídos formando un registro lógico. Este concepto puede faltar en la sentencia. La lista puede componerse de: nombres de variables, elementos de conjuntos, nombres de conjuntos, nombres de variables definidas como *carácter* y siempre separados unos de otros por comas (,).

Explicaremos todos estos conceptos con varios ejemplos.

a) `READ *, AA, BB, I`

Esta sentencia indica que van a leerse, dentro de la memoria, los campos o variables AA y BB (reales) e I (entero). El asterisco (*) que sigue a la palabra READ (leer), indica que el *formato* de esos datos será *generado por el* propio *ordenador* sin dejar opción al programador para suministrarlo. Este formato se conoce también con el nombre de *formato libre.*

Un programa que contenga esta sentencia, esperará que le sean suministrados esos tres datos y en ese orden. ¿A través de qué unidad de entrada han de ingresar esos datos? En general, desde el fichero o dispositivo externo designado por el propio sistema. Lo común es el teclado de un terminal o las fichas perforadas.

Los tres datos deberán escribirse o perforarse uno a continuación del otro, separados por una coma o por uno o varios blancos. Si el registro tecleado fuera, por ejemplo

```
35.587,– 570.283,56
```

el programa FORTRAN emparejaría estos valores con las variables correspondientes, del modo siguiente:

```
AA = 35.587
BB = –570.283
I = 56
```

Indicando por _ los blancos, también podrían haberse tecleado o perforado así:

```
35.587___ –570.283 ____ 56
```

E incluso podrían haberse perforado en tres fichas distintas, empleando una ficha para cada variable o campo.

Una vez más puede observarse ahora la diferencia entre registro físico y registro lógico. Cuando el programa FORTRAN encuentra la sentencia READ del ejemplo, lee el registro lógico formado por los tres campos AA, BB e I.

Estos tres campos pueden haber sido tecleados en un terminal pantalla-teclado o pueden haber sido perforados en una ficha. En ambos casos, la línea completa de la pantalla o la ficha, serían los registros físicos.

b) `READ 100, AA, BB, I(2)`

En este caso se leerán también tres valores AA, BB y el segundo elemento del conjunto I empleando los mismos medios descritos anteriormente dependiendo del sistema, pero en esta ocasión la constante entera 100 significa la etiqueta de una sentencia de formato, FORMAT, donde se describe la forma que tendrá cada variable: si es entera o real; cuántas posiciones ocupa. Dicha sentencia FORMAT deberá estar contenida en el mismo programa. Será estudiada más adelante. Con ella, el programador es libre de elegir y especificar los formatos que desea para sus variables. La constante entera 100 podría haberse indicado mediante una variable entera a la que se le hubiera asignado ese valor, por ejemplo N = 100, mediante una sentencia especial, ASSING, que será estudiada más adelante.

c) `READ '(2F8.2, I3)', AA, BB, I(2)`

Esta sentencia es análoga a la anteriormente descrita; pero en este caso los formatos de cada variable vienen indicados en la propia sentencia como una variable *carácter* y encerrados entre paréntesis. Aunque sea adelantar ideas,

```
2F8.2, I3
```

significa que tanto AA como.BB ocupan ocho posiciones, entre signo cifras y punto decimal, de las cuales dos de ellas forman la parte frac-

cionaria y que la variable entera I(2) ocupa tres posiciones cualquiera que sea el medio de entrada: ficha o teclado. Las letras F e I son formatos a emplear con variables reales o enteras, respectivamente.

d) Con la sentencia de entrada

```
READ *, NUM, NOM, SUEL
```

y los datos tecleados en un terminal así:

```
123 __ 'GUTIERREZ' __ 35423.85
```

el programa hubiera realizado la siguiente asociación de valores:

NUM = 123	variable entera (ocupa tres posiciones)
NOM = GUTIERREZ	variable carácter
SUEL = 35423.85	variable real (ocupa ocho posiciones con dos decimales)

e) Para el ejemplo anterior d), también podría haberse empleado como sentencia de entrada, la siguiente:

```
READ '(I3, A, F8.2)', NUM, NOM, SUEL
```

El efecto sobre los datos anteriores, hubiera sido el mismo.

A es un nuevo formato para indicar variables formadas por caracteres alfanuméricos, es decir, variables *carácter*.

La forma general de una sentencia PRINT es análoga a la READ básica, es decir,

```
PRINT f, lista
```

donde *f* y *lista* corresponden a los mismos conceptos que los definidos anteriormente, incluyendo ahora en la lista cualquier expresión FORTRAN (*).

Ejemplos:

a) `PRINT *, A, B, J`

(*) Existen compiladores en el mercado de ordenadores caseros donde las sentencias de E/S básicas, tienen la sintaxis READ (*, *) y WRITE (*, *), respectivamente.

Puesto que el formato de las cantidades que salen es controlado directamente por el ordenador, dependiendo de cuál sea la máquina empleada, así será ese formato. En general los datos enteros se escribirán como cantidades enteras (es el caso de la variable J) y los reales como números decimales con punto (caso de A y de B).

Con la sentencia anterior se perforan en una ficha, se imprimirán en una impresora, o se visualizarán en una pantalla, los contenidos de A B y J.

b) `PRINT *`

daría lugar a una tarjeta vacía o a una línea de pantalla o impresora completamente en blanco. No existe la lista de variables.

c) `PRINT 200, NUM, JOTA, A`

Escribir los contenidos de las variables NUM, JOTA y A de acuerdo con la sentencia de formato cuya etiqueta es 200.

d) `PRINT '(I3, F4.1, A)', J, AL, NOMB`

Escribir los contenidos de J, AL y NOMB, sabiendo que:

J es variable entera que ocupará tres posiciones.
AL es variable real con cuatro posiciones y un decimal
NOMB es variable *carácter* y por eso su formato es A.

e) `PRINT *, 'PARA _ RADIO _ = ', 8, 'LONGITUD = ', 321.7`

escribiría un registro, por ejemplo, como el siguiente:

`PARA _ RADIO _ = ___ 8 ___ LONGITUD = ___ 321.7 ·`

Insistimos en que el número de blancos que se producen entre los campos, dependerá del sistema empleado. Los espacios en blanco se han indicado con el símbolo _.

f) `PRINT '(A, I3, A, F6.1)', 'PARA_RADIO=', 8, '__LONGITUD =', 321.7`

generaría la siguiente salida:

`PARA_RADIO = __ 8 __ LONGITUD = _ 321.7`

g) `PRINT '(A, I3, A, F8.2)', ' PARA_RADIO = ', 8, '__ LONG = ', 8*3.14`

escribirá este registro:

`PARA_RADIO = __ 8 __ LONG = ___ 25.12`

El último campo, **25.12**, es el resultado de la expresión **8*3.14**.

Digamos para finalizar, que cada sentencia **READ** o **PRINT** que se ejecute en un programa, lee o escribe un nuevo registro, es decir, se lee una nueva línea de terminal o una nueva ficha o se escribe una nueva línea en la pantalla o impresora o se perfora una nueva ficha.

Con una misma sentencia **PRINT** puede escribirse una o más líneas, si los datos a imprimir no caben en una sola.

Programa 6. Se trata de leer diez valores de x cada uno de ellos aproximado hasta las décimas y siempre comprendidos entre ± 300.0. Según sea el valor de x, se calcula el correspondiente de y de acuerdo con las fórmulas:

si $0 \leqslant x \leqslant 100.0$ entonces $y = 2x^2 + 2x$

si $100.1 \leqslant x \leqslant 200.0$ entonces $y = 3x^2 + 5x$

Para cualquier otro valor de x, no generar ningún valor de y.

Deben leerse los valores de x y escribir en cada cálculo los de x e y.

El organigrama correspondiente sería el de la figura 3.6, en el cual I es un contador encargado de contabilizar el número de x que se leen.

Los números que figuran entre paréntesis en el organigrama, se corresponden con etiquetas de sentencias en la codificación FORTRAN, listada en la figura 3.7.

Comentarios al programa:

1. Para leer la variable **X** se ha utilizado una sentencia **READ** sin formato. Supongamos que el dato se introduce a través del teclado de un terminal.

2. La sentencia de etiqueta **80**, está escrita en dos líneas sucesivas y por ello la segunda lleva un **1** en la columna 6 de CONTINUACION.

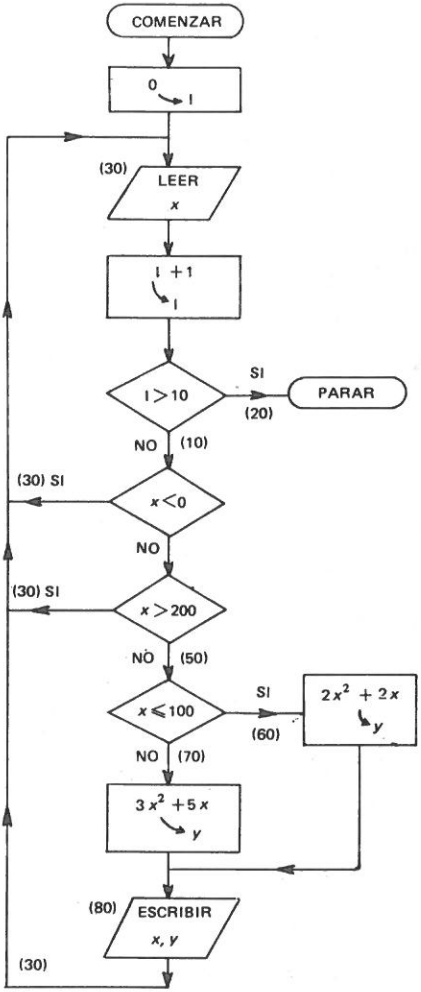

COMENZAR
0
I
(30)
LEER
x
I + 1
I
I > 10
SI
PARAR
(20)
NO
(10)
(30) SI
x < 0
NO
(30) SI
x > 200
NO
(50)
x ≤ 100
SI
2x² + 2x
y
(60)
NO
(70)
3x² + 5x
y
(80)
ESCRIBIR
x, y
(30)

Figura 3.6

```
      PROGRAM FUNC1
C
      I=0
   30 READ *,X
      I=I+1
      IF(I-10)10,10,20
   10 IF(X)30,40,40
   40 IF(X-200.)50,50,30
   50 IF(X-100.)60,60,70
   70 Y=3.0*X**2+5.0*X
   80 PRINT '(F6.1,F8.1)','VALOR DE X=',
     1X,' VALOR DE Y=',Y
      GO TO 30
   60 Y=2.0*X**2+2.0*X
      GO TO 80
   20 STOP
      END
```

Figura 3.7

3. La sentencia **PRINT** indica una salida con formato elegido por el programador:

   F6.1, seis posiciones para X que es real y un decimal
   F8.1, ocho posiciones para Y y un decimal

   La impresión de X vendrá precedida por la constante *carácter*

   VALOR _ DE _ X =

   y la de Y por

   _ VALOR _ DE _ Y =

4. Si la salida se produjera por el terminal, en la pantalla podría leerse, por ejemplo:

   VALOR_DE_X = __30.5 _ VALOR_DE_Y = __1921.5
   VALOR_DE_Y = _112.8 _ VALOR_DE_Y = _38735.5

   Este conjunto de dos registros escritos corresponde a dos valores de X leídos.

Programa 7. Se trata del mismo programa 6 pero ahora haciendo aplicación del DO y variando las formas de leer y escribir.

```
      PROGRAM FUNC2
C
      DO 10 I=1,10
        READ 100,X
        IF(X)10,20,20
   20   IF(X-200.0)30,30,10
   30   IF(X-100.0)50,50,60
   60   Y=3.0*X**2+5.0*X
   70   PRINT 200,X,Y
        GO TO 10
   50   Y=2.0*X**2+2.0*X
        GO TO 70
   10 CONTINUE
C
C  SENTENCIAS DE FORMATO
C
  100 FORMAT(F6.1)
  200 FORMAT('PARA X=',F6.1,'    Y='
     1,F8.1)
      STOP
      END
```

Figura 3.8

Comentarios al programa:

1. Los diez valores de X se leen mediante la sentencia

   DO 10 I = 1,10

   cuyo rango comienza en la sentencia siguiente READ 100, X y acaba en la sentencia

   10 CONTINUE

2. Para leer cada uno de los valores de X, se ha empleado una sentencia READ con formato. Esta sentencia significa: leer un valor de X de acuerdo con el formato contenido en la sentencia número 100.

3. La sentencia 100 FORMAT indica: leer un valor real (código F) que posee 6 posiciones en total, de las cuales una es decimal. Como ya hemos dicho anteriormente, en esas seis posiciones han de contarse los dígitos, el punto decimal y el signo si existiera.

4. Para escribir los resultados X e Y se emplea la sentencia de formato número 200 que está escrita en dos renglones sucesivos. Esta sentencia indica: escribir X con el formato F6.1 e Y con el F8.1, precedidos ambos valores de las constantes *carácter* 'RADIO =' y '_ _LONGITUD =', respectivamente.
5. Con este programa se producirán registros como éstos:

```
PARA_X = _ _30.5_ _ _ Y = _ _1921.5
PARA_X = _ 112.8_ _ _ Y = _ 38735.5
```

6. Como la variable X está comprendida entre los valores ± 300.0, se ha tomado para ella el formato F6.1.
   Como el resultado Y es siempre positivo (sin signo) y está aproximado también hasta las décimas, basta con el formato F8.1.

3.6. SENTENCIAS DE E/S PARA ACCESO SECUENCIAL

Como ya sabemos, las sentencias de E/S se utilizan para transmitir datos desde cualquier medio de entrada al almacenamiento principal o desde éste a cualquier dispositivo de salida.

Entre los dispositivos secuenciales de E/S figuran los terminales, las lectoras de fichas o de cinta magnética, las impresoras, las pantallas de los terminales, las perforadoras de ficha o cinta.

Existen dos sentencias de E/S para el acceso secuencial: READ y WRITE, cuyas formas sintácticas generales son, respectivamente:

```
READ (u, f, END = s, ERR = n, IOSTAT = v) lista
WRITE (u, f, END = s, ERR = n IOSTAT = v) lista
```

para información leída o escrita de acuerdo con un formato. O bien

```
READ (u, END = s, ERR = n, IOSTAT = v) lista
WRITE (u, END = s, ERR = n, IOSTAT = v) lista
```

para información leída o escrita sin formato.

Todas ellas son sentencias ejecutables y esas formas generales corresponden al FORTRAN completo. Se estudiarán con todo detalle en los párrafos siguientes desde las formas más simples a las más completas.

Otras sentencias auxiliares para el acceso secuencial y exclusivas para el manejo de cintas magnéticas, son:

END FILE	fin de fichero
REWIND	rebobinar
BACKSPACE	retroceder

3.7. SENTENCIA READ GENERAL

Ya se dijo en 3.4 al hablar de unidades de E/S que cada una de ellas posee una identificación para el ordenador. También se dijo que esa identificación varía de unos sistemas a otros. Matizando más, diremos que un fichero viene indicado en un programa FORTRAN, mediante una constante o variable entera de valor cero o positivo en ambos FORTRAN o por una expresión entera en el ampliado (*). Este número se denomina número de la *unidad lógica* en oposición a la *unidad física* que sirve de soporte al propio fichero.

Como números de identificación de las unidades de E/S, emplearemos para el desarrollo de nuestra teoría, entre otros, los siguientes:

1, para el terminal del usuario pantalla-teclado
2, para la impresora
3. para la unidad que designa el disco magnético
4, para una lectora/perforadora de fichas
5, para la unidad cinta magnética.

Ampliando ahora la definición de sentencia READ estudiada anteriormente, dando al usuario la posibilidad de que elija la unidad lógica que desee para leer sus datos, tendremos la siguiente nueva forma:

```
READ (u, f) lista
```

siendo: *u*, como ya se ha dicho más arriba en este mismo apartado, el número que designa la unidad de lectura.

(*) En *ficheros internos* (Véase APENDICE 3) puede emplearse como identificación: variable, conjunto, elemento de un conjunto o serie, todos ellos de tipo *carácter*.

Los demás parámetros se han definido anteriormente.

Veamos algunos ejemplos:

a) `READ (1, 100) A, B (3), KILO`

Indica la lectura de tres variables: A (real), el tercer elemento del conjunto B (real) y la variable (entera) KILO. Esa lectura se realizará a través de un terminal de usuario y de acuerdo con la instrucción de formato, FORMAT, número 100.

b) Con el par de sentencias

```
DIMENSION A(20)
READ (4, 300) A, B, C
```

escritas en un mismo programa, podrían leerse en memoria, de acuerdo con el formato 300 y por la unidad lectora de fichas, los siguientes campos: todo un conjunto A de variables reales, unidimensional, de veinte elementos; las variables también reales, B y C. El conjunto A, como ya se ha estudiado anteriormente, 2.6, se almacenaría siguiendo las reglas ya estudiadas.

c) Si tuvieramos una línea tecleada por el terminal, con el contenido

```
3.28, 40, –75.123
```

este registro podría leerse mediante la instrucción

```
READ (1, *) A, I, B
```

empleando para ello formato libre. Como resultado, sería:

```
A = 3.28
I = 40
B = –75.123
```

dentro de la memoria del ordenador.

d) Para el mismo fin que en c), podría haberse empleado la sentencia:

```
READ (1, '(F4.2, I2, F7.3)') A, I, B
```

Cuando se está leyendo un fichero de datos es muy útil poder saber dónde acaba, es decir, cuál es el último registro leído, la última ficha,

por ejemplo. Para ello se utiliza una sentencia READ, con un nuevo parámetro END, de la forma general siguiente:

```
READ (u, f, END = s) lista
```

Toda la información encerrada entre paréntesis y separada por comas, recibe el nombre de *lista de información de control.* El parámetro END = puede emplearse en ambos FORTRAN. La letra *s* significa la etiqueta de una sentencia ejecutable contenida en el mismo programa. El efecto de esta sentencia es: continuar leyendo el siguiente registro, si existe; en caso contrario bifurcar a la sentencia número *s*. La sentencia número *s* puede, por ejemplo, ordenar la impresión de un mensaje advirtiendo que la lectura del fichero ha acabado o indicar cualquier otra operación.

Por ejemplo:

```
READ (4, 300, END = 25) VELO, A, K
```

indica leer tres valores VELO, A, K por la unidad 4, de acuerdo con el formato indicado en la sentencia de etiqueta 300 y de manera que cuando el fichero de datos (fichas perforadas), finalice, se bifurque a la sentencia número 25.

Es posible escribir este tipo de sentencias empleando *palabras clave* para todos los parámetros así:

```
READ (UNIT=4, FMT = 300, END = 25) VELO, A, K
```

es decir, UNIT = para el número de unidad de entrada y FMT = para la etiqueta de formato.

Cuando en la lista de información de control se utilizan palabras clave, los parámetros contenidos en ella pueden escribirse en cualquier orden. Por ejemplo, la sentencia READ anterior es equivalente a las:

```
READ (FMT = 300, UNIT = 4, END = 25) VELO, A, K
READ (END = 25, UNIT = 4, FMT = 300) VELO, A, K
```

Cuando no se emplean palabras clave para unidad y formato, el órden es el establecido en la forma general antes descrita: primero unidad, luego formato.

Otra información de control que puede incluirse en esta sentencia READ, es la que posibilita al programador a controlar cualquier condición de error que se produzca durante una operación de lectura. Para ello se emplea la palabra clave ERR =. Con esta nueva opción la forma general de la sentencia READ sería:

```
READ (u, f, END = s, ERR = n) lista
```

siendo *n* la etiqueta de una sentencia ejecutable contenida en el mismo programa y cuyo efecto es producir una bifurcación a la misma de ocurrir un error de lectura. La sentencia número *n* puede, por ejemplo, ordenar la impresión de un mensaje advirtiendo del error.

Por ejemplo:

```
READ (1, 100, END = 10, ERR = 25) A, B, I
```

leería tres valores A, B, I por la unidad 1, de acuerdo con el formato descrito en la sentencia 100, bifurcando a la 10 al final del fichero o la 25 si se produce un error de lectura.

La sentencia anterior también podría haberse escrito así:

```
READ (END = 10, FMT = 100, UNIT = 1, ERR = 25) A, B, I
```

Por último, existe otra forma de comprobar el final de un fichero o la aparición de un error de lectura. Para ello se emplea la opción IOSTAT = ,del inglés Input Output Status, seguida de una variable entera o el nombre de un elemento de un conjunto entero.

Así:

```
READ (1, 200, END = 99, ERR = 98, IOSTAT = JOTA), A, B, C
```

leería las variables A, B, C, por la unidad 1, según el formato 200. La variable JOTA toma un valor determinado (>0) si se detecta un error, otro valor (<0) si se alcanza el fin de fichero y un tercero (=0) si la ejecución es correcta, además de producirse las bifurcaciones respectivas indicadas por 98 y 99. Consultar el manual de la instalación correspondiente para conocer esos valores de la opción IOSTAT.

El valor o contenido de JOTA puede ser comprobado por el programador, empleando las instrucciones adecuadas en el programa, como un IF o bien imprimiéndolo o visualizándolo en un terminal.

Veamos un ejemplo de aplicación de la opción IOSTAT. Sea un fichero secuencial contenido en una cinta magnética. Cada registro lógico consta de tres campos: NOMB, NUM y CUALI. Se desconoce el número de registros que componen el fichero y se pretende leerlos todos imprimiendo su contenido, uno a uno.

La codificación FORTRAN podría ser:

```
1  READ (5, 100, IOSTAT = IO) NOMB, NUM, CUALI
   IF (IO) 2, 3, 3
3  PRINT *, NOMB, NUM, CUALI
   GO TO 1
2  CONTINUE
   .
   .
   .
```

La variable IO es en este ejemplo cero, mientras no se llegue al final del fichero; es negativa cuando se alcance el mismo.

Por consiguiente, la forma más general de una sentencia READ es:

```
READ (u, f, END = s, ERR = n, IOSTAT = v) lista
```

siendo *v* una variable definida como entera.

En el FORTRAN básico únicamente pueden utilizarse en la lista de información de control, u, f y END=.

3.8. SENTENCIA WRITE

Esta sentencia se utiliza para transferir información desde la memoria principal a un soporte externo: impresora, terminal, perforadora de fichas, etc. Por consiguiente, podríamos decir que tiene una misión opuesta a la READ que acabamos de estudiar.

Es conveniente notar el número de caracteres que es posible escribir en algunos registros físicos clásicos:

Impresoras: normalmente 132 y en algunos casos 120 por línea

Terminales pantalla-teclado: hasta 80 caracteres/línea.

Fichas: 80 caracteres/ficha

Todo lo dicho para la sentencia READ podría repetirse para la WRITE. Por ello, no lo vamos a señalar una vez más aquí. Las formas generales de una sentencia WRITE con formato son:.

```
WRITE (u, f) lista

WRITE (u, f) lista

WRITE (u, f, ERR = n) lista

WRITE (u, f, ERR = n, IOSTAT = v) lista
```

Veamos algunos ejemplos:

a) `WRITE (2,100)`

Esta sentencia no tiene *lista*. Significa imprimir de acuerdo con el formato contenido en la sentencia número 100. Este tipo de sentencia se emplea para imprimir títulos o cabeceros contenidos en el FORMAT.

b) `WRITE (1, 100) A, B (1), B (5), I`

Escribe las variables A, el primer y quinto elemento del conjunto B y la I. Medio empleado: el terminal. Formato: el definido en la sentencia 100.

c)
```
DIMENSION A(3,4)
WRITE (2, 200, ERR = 25) A, B
```

Con estas dos sentencias se imprimiría además de la variable real B, todo el conjunto bidimensional A de doce elementos y ellos por columnas (ver 2.6), es decir, A(1,1), A(2,1), A(3,1), A(1,2), ... Si se detecta algún error de salida, se bifurcaría a la sentencia número 25.

Programa 8. Se posee un bloque de registros, cada uno de ellos en una ficha perforada. Cada registro contiene la edad de una persona expresada en años, como un número entero de un máximo de dos cifras y una clave, también número entero.

La clave es igual a 1 cuando el registro corresponde a varón y 2 a hembra. Se desconoce el número total de registros o fichas. Calcular la edad media de los varones y de las hembras. Hacer uso de los parámetros END = y ERR =.

Se designan por IEDAD e ICLA las variables edad y clave, respectivamente. Cuando se trata de varón, las edades se acumulan para hallar la media en IEDADV. Cuando se trata de mujer, en IEDADH. El número total de varones y hembras se encuentra en NTOTV y NTOTH, respectivamente. Las medias se indican por MEDIAV y MEDIAH. Todas las cantidades son enteras.

El organigrama sería el representado en la figura 3.9.

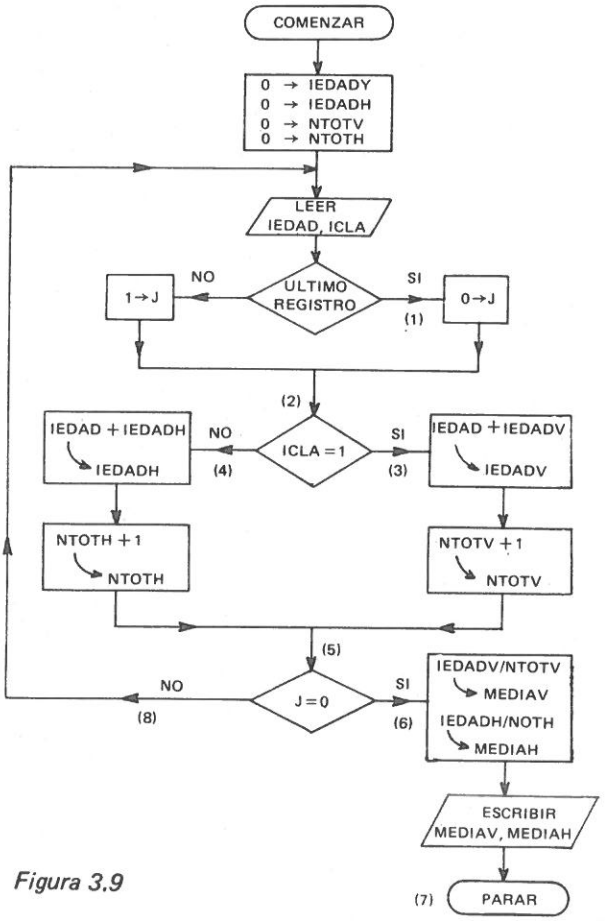


Figura 3.9

Teniendo en cuenta que se leen fichas por la unidad 4 y se escriben los resultados por la impresora, unidad 2, el programa FORTRAN podría ser el listado en la figura 3.10.

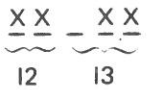
```
      PROGRAM MEDIAS

      IEDADV=0
      IEDADH=0
      NTOTV=0
      NTOTH=0
    8 READ(4,100,END=1,ERR=7)IEDAD,ICLA
      J=1
      GO TO 2
    1 J=0
    2 IF(ICLA-1)4,3,4
    3 IEDADV=IEDADV+IEDAD
      NTOTV=NTOTV+1
      GO TO 5
    4 IEDADH=IEDADH+IEDAD
      NTOTH=NTOTH+1
    5 IF(J)8,6,8
    6 MEDIAV=IEDADV/NTOTV
      MEDIAH=IEDADH/NTOTH
      WRITE(2,101)MEDIAV,MEDIAH

  FORMATOS DE E/S

  100 FORMAT(I2,I1)
  101 FORMAT(I2,I3)
    7 STOP
      END
```

Figura 3.10

Comentarios al programa:

1. Se ha introducido una variable de control, la J, para detectar el último registro leído, con el cual de todas formas habrá que realizar los cálculos correspondientes. Cuando sea J = 0, se calcularán los valores medios y se finaliza el programa.
2. En el programa FORTRAN, todas las variables se inicializan asignándoles el valor *cero*.
3. Cuando se trate del último registro, se bifurca a la instrucción número 1, donde se hace precisamente J = 0. Si existe un error de lectura de datos, el programa se detiene.
4. La sentencia de formato número 100, indica que se leerán los valores enteros que ocupan, respectivamente, *dos* y *una* columnas en la ficha perforada.

5. El formato número 101, indica que se imprimirán dos variables enteras, cada una de ellas ocupando dos columnas o posiciones. El hecho de haber asignado 13 a la segunda variable ha sido para dejar un espacio en blanco entre ellas. Si indicamos por X cualquier dígito y con _ las posiciones de impresión, para los valores más altos de las medias, sería:



3.9. SENTENCIAS READ Y WRITE SIN FORMATO

Ya dijimos en 3.6 que existían sentencias de E/S sin formato, aplicables a ficheros de acceso secuencial. Veamos ahora cuáles son estas sentencias y para qué se emplean.

La sentencia

```
WRITE (6) A, B, I
```

da lugar a que se escriban las variables A, B, I sobre el medio auxiliar (una cinta magnética) número 6. Todo ello sin definición de formato. Eso indica que la información pasará directamente desde la memoria principal a la memoria auxiliar tal como está, es decir, sin conversión alguna o, lo que es lo mismo, en binario (ceros y unos).

Este tipo de sentencia es útil cuando el programador lo que pretende es almacenar provisionalmente unos datos que luego volverá a emplear, es decir, volverá a leer en memoria principal, utilizando la sentencia correspondiente

```
READ (6) A, B, I
```

que se encarga de leer las mismas variables escritas A, B e I desde la misma unidad, la 6, con el formato binario que tengan, ya que no se indica otro. Cuando se utiliza este tipo de sentencias es porque se supone que el programador no va a precisar nunca leer o interpretar la información correspondiente.

Esta forma de leer o escribir, se denomina *sin formato* o bien en *formato legible* únicamente *para la máquina.* Los registros escritos o leídos de esta forma, se dice que son *registros sin formato.*

En este tipo de sentencia, pueden también utilizarse los parámetros END=, ERR= e IOSTAT=. Al número de la unidad podría precederle el parámetro UNIT=. Con todo ello, las sentencias de E/S, en modalidad secuencial y sin formato (formato binario) serían:

```
READ (u, END = s, ERR = n, IOSTAT = v) lista
```

```
WRITE (u, END = s, ERR = n, IOSTAT = v) lista
```

3.10. SENTENCIAS END FILE, REWIND y BACKSPACE

Ya anticipamos que, además de las sentencias propias de E/S, existen otras auxiliares que sirven para manipular los ficheros secuenciales sobre cintas magnéticas.

Estas sentencias son las siguientes:

1. Sentencia para *fin de fichero.* Su forma más simple es:

```
END FILE u
```

siendo *u* una constante o variable entera que indica el número de un dispositivo donde se encuentra la cinta magnética. Puede estar precedido de UNIT= en el FORTRAN general.

La sentencia da lugar a que se grabe en la cinta correspondiente con un registro o marca de fin de fichero (Véase 3.4).

En esta sentencia pueden también incluirse otros parámetros como END=, ERR=, IOSTAT=, dando entonces lugar a la forma general siguiente:

```
END FILE (UNIT = u, END = s, ERR = n, IOSTAT = v)
```

Ejemplos:

```
END FILE 9
END FILE (9, END = 14, ERR = 16)
END FILE (UNIT = 9, END = 100, ERR = 200, IOSTAT = MAT)
```

No tiene sentido emplear la opción IOSTAT =, sin la ERR =, ya que de producirse un error al grabar una marca de fin de fichero, la señal o valor correspondiente se almacenará en la variable *v* (MAT en nuestro último ejemplo) deteniéndose el programa. Es necesario tener la posibilidad de bifurcar a la sentencia que indica ERR = n para detectar tal anomalía y conocer el contenido de IOSTAT.

Esta sentencia también puede escribirse así: ENDFILE

2. Sentencia para *rebobinar* una *cinta*. Su forma más simple es:

```
REWIND u
```

la cual da lugar a que el fichero colocado en el dispositivo *u* se posicione de tal manera que el siguiente registro a leer en memoria principal sea el primero del fichero. Puede estar precedido por UNIT =, en el FORTRAN general.

Su forma ampliada es:

```
REWIND (UNIT = u, ERR = n, IOSTAT = v)
```

3. Sentencia para hacer *retroceder la cinta*. Su forma más sencilla es:

```
BACKSPACE u
```

siendo *u* una constante o variable entera que indica que el fichero situado en el dispositivo *u* se posicione de tal manera que el siguiente registro a leer sea precisamente el que acaba de leerse; es decir, se re-

trocede un registro completo. Puede venir precedido por UNIT =, en el FORTRAN general.

La forma ampliada será:

```
BACKSPACE (UNIT = u, ERR = n, IOSTAT = v)
```

Con el FORTRAN básico únicamente pueden emplearse las formas más simples de estas sentencias.

El uso de estas sentencias REWIND, BACKSPACE y END FILE es privativo del acceso secuencial, es decir, de los fichéros SAM.

Con el FORTRAN completo, *u* puede ser también cualquier expresión entera, real o doble precisión. Pero esa expresión siempre tomará el valor entero más próximo por defecto o cero.

3.11. SENTENCIAS DE E/S PARA ACCESO DIRECTO

Ya hemos dicho anteriormente, que el acceso directo o acceso al azar únicamente se consigue utilizando discos magnéticos, diskettes o, aunque menos usados, tambores magnéticos (Fig. 3.3)

Para manejar estos medios de memoria auxiliar, FORTRAN 77 dispone de las sentencias READ y WRITE correspondientes.

Las formas generales para el FORTRAN completo son:

```
READ (u, f, REC = r, ERR = n, IOSTAT = v) lista
```

```
WRITE (u, f, REC = r, ERR = n, IOSTAT = v) lista
```

donde REC = es un nuevo parámetro de la lista de opciones en el cual *r* es una expresión entera mayor que cero que representa el número del registro a partir del cual va a leerse o escribirse la información, es decir, las variables que figuran en la lista. Este parámetro no puede emplearse nada más que en el acceso directo, es decir, ficheros creados como DAM.

Las sentencias anteriores utilizan control de formato *f*.

Los valores *u* y *f* pueden estar precedidos por UNIT = y FMT =, respectivamente.

Por ejemplo:

```
READ (3, 100, REC = 20, ERR = 103) A(1), A(2), A(3)
```

significa leer las variables A (1), A (2) y A (3), de acuerdo con el formato número 100, que están situadas en la mitad lógica 3 a partir del registro cuyo número es el 20. Si se produce error en la lectura, bifurcar a la sentencia número 103 del mismo programa. La opción ERR = no puede emplearse en el FORTRAN básico.

Para lectura o escritura sin formato, las sentencias generales son:

```
READ (u, REC = r, ERR = n, IOSTAT = v) lista
```

```
WRITE (u, REC = r, ERR = n, IOSTAT = v) lista
```

Lo mismo que se dijo al hablar de acceso secuencial sin formato, toda lectura sin formato, es decir, en binario, desde el disco a la memoria principal tiene que haber sido precedida por otra sentencia de escritura, también sin formato, desde la memoria principal a la secundaria o disco magnético.

Por último, debe observarse que un fichero sobre un dispositivo de acceso directo, como es el disco, puede procesarse como SAM o como DAM.

Ejemplos válidos con formato serían:

```
READ (3, FMT = 500, REC = 50, IOSTAT = JJ) KILO, VAR
READ (UNIT = 3 * I, FMT = 100, REC = 23) A, B, C, D22
WRITE (FMT = 200, REC = 55, UNIT = 3)  A(3), B, C(5)
WRITE (3, '(F3.2)', REC = 5)
```

Otros ejemplos válidos de sentencias sin formato son:

```
READ (3, REC = 80, ERR = 100) A, X, Y
WRITE (REC = 80, UNIT = 3, IOSTAT = MAX) Z, Y, W, A (1)
```

3.12. DO IMPLICITO EN LISTAS DE E/S

Según lo que hemos visto hasta aquí, en las sentencias de E/S puede figurar una relación o lista de variables a leer o a escribir. Recordando los ejemplos, esa lista podrá estar formada por variables, simples, es decir, no subindicadas; por variables con subíndices o por nombres de conjuntos (Véase 3.7, ejemplo b).

Volviendo ahora al caso de un conjunto A de treinta elementos, por ejemplo, con los conocimientos que tenemos hasta aquí, podría leerse de varias formas distintas:

a) Mediante el par de sentencias

```
DIMENSION A (30)
.....
.....
READ (4, 100) A
```

b) Empleando una sentencia DO de este modo

```
      DIMENSION A(30)
      DO 3 I = 1,30
    3 READ (4,100) A (I)
```

donde el rango se repite treinta veces, leyéndose los treinta valores distintos del conjunto A, pero cada uno de un registro físico o ficha perforada, ya que cada ejecución de la sentencia READ o WRITE genera la lectura o escritura de un registro físico diferente.

c) Si considerásemos que en cada una de las fichas anteriores pudieran figurar seis valores del conjunto A, entonces otra forma de introducir en memoria dicho conjunto, economizando registros físicos sería:

```
      DIMENSION A(30)
      DO 5 I = 1,25,6
    5 READ (4,200) A (I), A(I + 1), A(I + 2), A(I + 3), A(I + 4), A(I + 5)
```

El FORTRAN 77 dispone de una forma más sencilla orientada a casos como los descritos aquí. Es el uso del DO *implícito* cuya forma general, para el caso de un conjunto unidimensional, es:

(N conj (i), i = m_1, m_2, m_3)

siendo: *Nconj*, la variable entera o real que designa el nombre del conjunto.
i, variable entera en el FORTRAN básico o entera, real o doble precisión en el completo.
m_1, m_2 y m_3, son constantes o variables enteras en el FORTRAN básico o expresiones enteras, reales o doble precisión en el completo que indican, respectivamente, un valor inicial, final e incremento de variación.

Obsérvese que el DO implícito sigue las reglas de la sentencia DO (véase 2.12). La coma (,) entre el nombre del conjunto e *i* es obligatoria. Cuando es $m_3 = 1$, este valor puede omitirse. Todo el DO implícito está encerrado entre paréntesis.

Ejemplos:

a) `READ (4, 100, END = 83) (A(I), I = 3, 8, 1)`

significa la lectura de las variables

A(3), A(4), A(5), A(6), A(7) y A(8)

b) `WRITE (4,200) (ALFA (J), J = 1, 6, 2)`

significa que se escribirán los elementos

ALFA(1), ALFA(3) y ALFA(5)

del conjunto ALFA.

c) `READ (N, 500, ERR = 35) (A(I), B(I), C(I), I = 1,3)`

da lugar a la lectura de elementos de distintos conjuntos y en el orden siguiente:

A(1), B(1), C(1), A(2), B(2), C(2), A(3), B(3) y C(3)

d) `WRITE (M + 3, 100) A, C(3), (X(I), I = 2,7,2)`

escribiría las variables siguientes:

A, C(3), X(2), X(4) y X(6)

El DO implícito también puede emplearse para leer o escribir conjuntos con más de una dimensión. Veamos el significado del ejemplo siguiente:

```
READ (N, 100) ((A (I, J), I = 1, 3), J = 1,2)
```

En este ejemplo se lee una serie de datos o campos por la unidad N de acuerdo con la sentencia de formato número 100. Esos datos forman un conjunto bidimensional A. En la sentencia anterior existen dos niveles de DO implícitos: el primero o más interior es el

```
(A (I, J), I = 1,3)
```

y el segundo o más exterior, el

```
((1er nivel), J = 1,2)
```

Constituyen dos sentencias DO en nido. Obsérvese que cada nivel está encerrado entre paréntesis y uno está separado del otro por una coma (,).

El significado de esta sentencia es: la variable J toma el valor 1, variando entonces la I desde 1 hasta 3 de uno en uno. A continuación J toma el valor 2 y la I vuelve a variar desde 1 hasta 3 de uno en uno, finalizando así el proceso. Con todo ello se leerá la siguiente lista de campos y en este orden:

```
A (1,1), A (2,1), A (3,1), A (1,2), A (2,2) y A (3,2)
```

Resumiendo: por cada valor que toma el subíndice exterior (el J), el interior (el I), toma todos los posibles.

El empleo del DO implícito puede extenderse a más de dos dimensiones. Veamos un último ejemplo para un conjunto de tres dimensiones:

```
WRITE (M, 250, END = 25, ERR = 10) (((A (I, J, K), I= 1,2),
J = 1,2), K = 1,3)
```

Esta sentencia daría lugar a la siguiente salida en escritura:

```
A (1,1,1), A (2,1,1), A (1,2,1), A (2,2,1), A (1,1,2), A (2,1,2), A (1,2,2)
A (2,2,2), A (1,1,3), A (2,1,3), A (1,2,3) y A (2,2,2)
```

Revisemos ahora la relación entre el DO implícito y la forma de almacenarse en memoria los conjuntos de dos dimensiones. Por extensión podrá deducirse la misma técnica para conjuntos con más dimensiones. Al estudiar los conjuntos bidimensionales y su disposición en memoria (véase 2.6) se dijo que éstos se almacenan siguiendo el orden de columnas, es decir, variando más rápido el primer subíndice. Entonces la disposición en memoria de un conjunto A de dos filas y tres columnas, es decir:

```
A (1,1)   A (1,2)   A (1,3)
A (2,1)   A (2,2)   A (2,3)
```

sería la siguiente:

VARIABLE	A(1,1)	A(2,1)	A(1,2)	A(2,2)	A(1,3)	A(2,3)
UBICACION	1	2	3	4	5	6

Se supone que A (1,1) ocupa la primera posición de memoria, A (2,1) la segunda, etc.

Supongamos que dicho conjunto A se lee mediante la sentencia

```
READ (N, 100)  ((A (M,N), M = 1,2), N = 1,3)
```

Entonces la secuencia de lectura o secuencia de entrada de variables, sería:

ORDEN DE ENTRADA	1°	2°	3°	4°	5°	6°
VARIABLE	A(1,1)	A(2,1)	A(1,2)	A(2,2)	A(1,3)	A(2,3)
UBICACION	1	2	3	4	5	6

Es decir, en este caso tanto el orden en que van entrando en memoria las variables como la ubicación, coinciden: las variables se leen y se almacenan siguiendo el orden de las columnas.

Pero si para leer el mismo conjunto empleásemos la sentencia:

```
READ (N,100) ((A (M,N), N = 1,3), M = 1,2)
```

entonces la secuencia de entrada sería diferente y tal como se indica a continuación:

ORDEN DE ENTRADA	1º	4º	2º	5º	3º	6º
VARIABLE	A(1,1)	A(2,1)	A(1,2)	A(2,2)	A(1,3)	A(2,3)
UBICACION	1	2	3	4	5	6

En este caso, aunque las variables quedan, al final, almacenadas por columnas, el orden en el que van entrando no coincide con el de almacenamiento.

3.13. LA INSTRUCCION OPEN

En 3.4. se ha definido ya lo que es abrir y cerrar un fichero. Ahora estudiaremos las órdenes de que dispone el lenguaje FORTRAN 77 para realizar estas dos operaciones.

Hemos visto y empleado también números enteros para designar las unidades externas donde se almacena la información, y aunque tales números dependen y cambian de unos sistemas a otros, nosotros hemos empleado figurativamente el 1 para el terminal de usuario; el 4 para la lectora/perforadora de fichas, etc. (véase 3.7). Pero supongamos ahora, que nuestro sistema tuviera dos unidades de cinta, designadas por los números 5 y 6. Esto significaría que nuestro programa FORTRAN podría acceder a dos unidades de este tipo indistintamente. Ahora bien; en cada cinta magnética, y lo mismo ocurriría con el disco magnético, normalmente siempre existen varios ficheros. Por consiguiente, es necesario disponer de un medio para saber, en cada momento, qué fichero desea utilizarse. Este hecho se denomina *conectar* un fichero específico a una unidad, con lo cual quedará accesible para su uso, de manera que cualquier instrucción de lectura o escritura que se ordene posteriormente, se efectuará ya refiriéndose al número de la unidad correspondiente.

Resumiendo: conectar es sinónimo de abrir y ello se realiza en FORTRAN 77 empleando la instrucción OPEN, cuya forma general es

```
OPEN (lista de opciones)
```

indicando *lista de opciones*, una serie de características relacionadas con el fichero a conectar y que se describen a continuación. Además en el cuadro 3.1 se han recogido todas ellas con su objetivo. (Pág. 149).

En versiones anteriores del lenguaje FORTRAN, se disponía de la instrucción DEFINE FILE para esta misma misión.

Las opciones son las siguientes:

1. UNIT = u, donde *u* es una expresión entera que indica la unidad de entrada/salida a la que debe conectarse el fichero. Este parámetro tiene que figurar siempre y precedido o no de la palabra UNIT =. Si esta palabra no figura, *u* tiene que escribirse en primer lugar; en caso contrario, puede escribirse en cualquier posición dentro de la lista de opciones.

   Las opciones que siguen pueden figurar o no. Veamos cuales son.

2. ERR = n, que tiene el mismo significado que el ya estudiado en las sentencias de E/S.
3. IOSTAT = v. Ver sentencias de E/S.
4. STATUS = est, siendo *est* una expresión de tipo *carácter*, y por consiguiente encerrada entre comas, que toma cualquiera de los siguientes valores cuyos significados se indican:

'NEW'	nuevo
'OLD'	viejo
'SCRATCH'	de trabajo
'UNKNOW'	desconocido

   Si esta opción se omite, se considera el último valor: desconocido o 'UNKNOW'.

   Expliquemos cada uno de estos valores.

   a) NEW indica que el fichero con el que se va a trabajar no existía con anterioridad y por consiguiente se va a *crear* a continuación. En este caso, la información se transmitirá desde la memoria principal al fichero.

b) OLD indica que se va a hacer uso de un fichero que ya existía sobre la cinta o el disco: es una *explotación* del fichero mediante la cual la información pasa desde él a la memoria principal.

c) SCRATCH significa que el fichero se va a emplear temporalmente como medio de trabajo, de manera que una vez finalizado el programa, el fichero puede desaparecer. En este caso no tiene que figurar el nombre de dicho fichero en la lista de opciones. (Ver a continuación 5. FILE = nom).

d) UNKNOW significa que en este caso el usuario o programador desconoce el nombre del fichero y será el propio sistema el que busque en el índice de ficheros de cuál se trata descubriendo además el estado: nuevo, viejo. Si ya existía, su estado será OLD; en caso contrario será creado con estado NEW.

Cuando un fichero se abre con estado OLD, tiene que haber sido creado con anterioridad, ya que si no ocurriría un error. El error también ocurre cuando se pretende crear un fichero al que se le atribuye un nombre que ya pertenece a otro anteriormente creado.

Si STATUS no figura en la lista, el sistema le atribuye UNKNOW.

5. FILE = nom, siendo *nom* el nombre adjudicado al fichero. Es una constante *carácter* encerrada, por consiguiente, entre comas. Corresponde al nombre del fichero que va a conectarse con la unidad o dispositivo indicado por *u*.

6. ACCESS = acce, siendo *acce* una expresión *carácter* que indica, si el acceso al fichero será secuencial o directo. Para ello se emplean, respectivamente, las palabras 'SEQUENTIAL' o 'DIRECT'.

Cuando esta opción no figura en la lista, se entiende que el acceso es secuencial.

Veamos ahora algunos ejemplos con las opciones que acaban de describirse.

a) OPEN (5, FILE = 'PRIMAS')

Abre el fichero de nombre PRIMAS sobre la unidad 5. Su estado es desconocido y será determinado por el sistema. Su

acceso será secuencial por no figurar la opción correspondiente.

b) OPEN (5, IOSTAT = N (2), FILE = 'SUELDO', STATUS = 'OLD', ACCESS = 'SEQUENTIAL')

Abre el fichero secuencial SUELDO sobre la unidad de cintas número 5. Se trata de un viejo fichero externo.

c) OPEN (FILE = 'NOMINA', UNIT = IN + 2, STATUS = 'OLD', ERR = 100)

Abre el fichero NOMINA que es secuencial. Se trata de un fichero ya creado puesto que es OLD y que se encuentra sobre la unidad IN + 2 que depende del valor adjudicado a la variable IN. Si ocurre un error, bifurcar a la sentencia número 100.

Continuemos ahora describiendo algunas opciones más.

7. FORM = form, siendo *form* una expresión *carácter* con estas dos posibilidades:

'FORMATTED'	para información con formato.
'UNFORMATTED'	para información sin formato o en binario.

Cuando esta opción no figura en la lista, el sistema considera FORMATTED para los ficheros de acceso secuencial y UNFORMATTED para los de acceso directo.

8. BLANK = bl, siendo *bl* una expresión *carácter* con dos posibilidades:

'ZERO', con la cual cualquier carácter blanco se interpretará como cero. (Véase 4.12).

'NULL', mediante la cual los blancos se ignoran, a menos que un campo esté completamente lleno de blancos en cuyo caso se interpretará como un campo de ceros. Esta opción únicamente puede emplearse con información que esté sometida a un formato.

Para los ficheros de acceso directo aun existe otra opción que es la siguiente:

9. RECL = lgr, siendo *lgr* una expresión entera que indica la longitud de cada registro del fichero al que se va a acceder directamente. La longitud se expresa en número de caracteres si el fichero se define con un formato y en *bytes* si no tiene propia instalación donde vaya a trabajarse al emplearse esta opción.

Veamos ahora, algún ejemplo más.

a) OPEN (M + 2, FILE = 'XXX', ACCESS = 'DIRECT',
FORM = 'FORMATTED', RECL = 100, STATUS = 'OLD')

Abre el fichero de nombre XXX sobre la unidad M+2, al que se va a acceder de forma directa. Se trata de un fichero ya creado anteriormente en el que sus registros tienen una longitud de cien caracteres. Está sometido a un formato.

b) OPEN (IOSTAT = IN, ERR = 80, UNIT = IN * 2, RECL = 32,
ACCESS = 'DIRECT')

Abre un fichero sobre disco en la unidad IN * 2, Como no figura la opción FORM = será sin formato, y la longitud de cada registro será de 32 *bytes*.

Por último, y en relación con esta sentencia, digamos que las únicas opciones admitidas en el FORTRAN básico son:

u
ACCESS = 'DIRECT'
RECL = lgr

Con todo lo dicho se entiende que, una sentencia OPEN puede emplearse para:

- Conectar un fichero ya existente a una determinada unidad.
- Crear un fichero que ha sido preconectado con anterioridad.
- Crear un fichero y conectarlo a una unidad.
- Cambiar ciertos especificadores en un fichero conectado a una unidad.

PARAMETRO	COMENTARIOS
UNIT = expresión entera	Indica la unidad de E/S a la que estará conectado el fichero. UNIT puede faltar.
ERR = etiqueta de sentencia	Bifurcar si existe error.
IOSTAT = variable entera	Control de estado.
STATUS = expresión *carácter*	Opciones: 'NEW', 'OLD', 'SCRATCH', 'UNKNOW'. Si no figura opción, se considera la 'UNKNOW'.
FILE = expresión *carácter*	Nombre del fichero.
ACCESS = expresión *carácter*	Opciones : 'SEQUENTIAL', 'DIRECT'. Si no figura, se considera la 'SEQUENTIAL'
FORM = expresión *carácter*	Opciones: 'FORMATTED', 'UNFORMATTED'. Cuando no figura, se considera 'FORMATTED' para ficheros secuenciales y 'UNFORMATTED' para los de acceso directo.
BLANK = expresión *carácter*	Opciones: 'ZERO' para hacer equivalentes los blancos a los ceros y 'NULL' para ignorar los blancos.
RECL = expresión entera	Longitud de cada registro. Solo acceso directo.

Cuadro 3.1

3.14. LA INSTRUCCION CLOSE

Esta sentencia rompe la conexión de un fichero con una unidad. No figura el FORTRAN 77 básico. Digamos que existen varias aplicaciones donde es necesario emplear dicha sentencia:

- Al final de la ejecución de un programa puesto que el fichero no va a continuar utilizándose.
- Cuando el fichero va a ser abierto o conectado posteriormente pero de otra forma, es decir, cambiando parámetros de definición.

– Cuando la unidad donde reside va a ser conectada a otro fichero.
– Cuando se desea conectar un fichero a una unidad diferente.

La forma general de esta sencia es

```
CLOSE (lista de opciones)
```

La *lista de opciones* contiene las siguientes: número de la unidad *u* que tiene que ser precedido por UNIT = cuando no figura en primer lugar; ERR = n e IOSTAT = v (ambas opcionales). Otra opción es la STATUS = s, siendo *s* una expresión *carácter* con dos posibilidades: dades:

'KEEP'	(guardar) que significa que el fichero debe retenerse o guardarse aun después de finalizado el programa. Esta opción, naturalmente, no puede utilizarse con ficheros del tipo temporal o SCRATCH descritos en 3.13.
'DELETE'	(borrar) que significa que el fichero debe borrarse al finalizar el programa.

Si no figura ninguna de estas dos posibilidades de la opción STATUS = se considera la 'KEEP'.

Ejemplos:

a) CLOSE (5, STATUS = 'DELETE')

Cierra el fichero sobre la unidad número 5, borrándolo.

b) CLOSE (M, ERR = 10, IOSTAT = IN, STATUS = 'KEEP')

Cerrar el fichero de la unidad M, conservándolo.

Ya se comprende que el hecho de cerrar una unidad que no existe o bien que no tiene ningún fichero conectado a ella no causa efecto alguno pero además tampoco produce error.

Una unidad puede cerrarse en un programa FORTRAN y volver a abrirse en el mismo programa; también puede ser conectada posteriormente al mismo u otro fichero distinto. De la misma forma, un fichero

que ha sido desconectado de una unidad mediante la instrucción CLOSE, puede abrirse con OPEN posteriormente, a la misma o diferente unidad.

Los ficheros temporales o SCRATH son automáticamente borrados al cerrarlos.

Los definidos como 'OLD' o 'NEW', son conservados y archivados en un determinado medio de memoria externa.

3.15 LA INSTRUCCION INQUIRE

Esta instrucción únicamente figura en el FORTRAN 77 completo. Con ella puede preguntarse al sistema determinadas propiedades de un fichero particular cuyo nombre se conoce o del que se sabe cuál es la unidad a la que está conectado.

La sentencia INQUIRE, puede figurar en un programa y por consiguiente ejecutarse, antes, mientras o bien después de que el fichero haya sido conectado a una unidad.

De acuerdo con lo dicho anteriormente, la sentencia INQUIRE tendrá dos formas:

a) Preguntar por las características de un fichero cuyo nombre *nom* es conocido:

```
INQUIRE (FILE = nom, lista de especificadores)
```

b) Preguntar por las características de un fichero del que se conoce la unidad *u* a la que está conectado:

```
INQUIRE (UNIT = u, lista de especificadores)
```

Suele decirse que se trata de una instrucción INQUIRE por fichero o por unidad, respectivamente, o mejor, *por nombre de fichero* o *por número de unidad.*

La *lista de especificadores* está formado por una o varias de las opciones que figuran en los cuadros 3.2, 3.3, 3.4 y 3.5 en los que además se indica el significado de cada uno de ellos. También pueden incluirse

	ESPECIFICADOR	RESULTADO ASIGNADO A LA VARIABLE AL PREGUNTAR	
		POR FICHERO	POR UNIDAD
	EXIST = variable lógica	Si el fichero existe, · TRUE · Si no existe, · FALSE ·	Si la unidad existe, · TRUE · Si no existe, · FALSE ·
SI EL FICHERO EXISTE	OPENED = variable lógica.	Si el fichero está conectado a la unidad, · TRUE · Si no está conectado, · FALSE ·	Si la unidad está conectada al fichero, · TRUE · Si no está conectada, · FALSE ·
	NAMED = variable lógica	Si el fichero tiene un nombre *nom*, el valor será, · TRUE · Si no lo tiene, · FALSE ·	Si el fichero conectado a la unidad tiene un nombre, · TRUE · Si no lo tiene, · FALSE ·
	NAME = variable *carácter*	Se le asigna el valor de *nom* (nombre del fichero). La sentencia de asignación *carácter* será estudiada en otro capítulo.	Se le asigna el nombre *nom* del fichero conectado si la unidad está conectada a un fichero con nombre. En caso contrario queda indefinido.
	SEQUENTIAL = variable *carácter*	Se le asigna el valor 'YES' si el fichero puede ser conectado como SAM. Se le asigna el valor 'NO' si no puede abrirse como fichero SAM. Se le asigna 'UNKNOWN' si no es posible determinar si el fichero puede conectarse o no como SAM.	Idem que *por unidad.*
	DIRECT = variable *carácter*	Se le asigna el valor 'YES' si el fichero puede ser conectado como DAM. Se le asigna el valor 'NO' si no puede abrirse como fichero DAM. Se le asigna 'UNKNOWN' si no es posible determinar si el fichero puede conectarse o no como DAM.	Idem que *por unidad.*
	FORMATTED = variable *carácter*	Se le asigna 'YES', si el fichero puede conectarse como fichero con formato. Se le asigna 'NO' en caso contrario. Se le asigna 'UNKNOWN' en los demás casos.	Idem que *por unidad.*
	UNFORMATTED = variable *carácter*	Se le asigna 'YES', si el fichero puede conectarse como fichero sin formato. Se le asigna 'NO' en caso contrario. Se le asigna 'UNKNOWN' en los demas casos.	Idem que *por unidad.*

Cuadro 3.2

	ESPECIFICADOR	RESULTADO ASIGNADO A LA VARIABLE AL PREGUNTAR	
		POR FICHERO	POR UNIDAD
SI EL FICHERO EXISTE Y ESTA CONECTADO	NUMBER = variable entera	Se le asigna el valor de la unidad externa conectada al fichero.	Idem que *por unidad*.
	ACCESS = variable *carácter*	Si el fichero está conectado como SAM, se le asigna 'SEQUENTIAL'. Si está conectado como DAM, se la asigna 'DIRECT' En otros casos 'UNKNOWN'.	Si la unidad está conectada a un fichero SAM, se le asigna 'SEQUENTIAL'. Si está conectado a un fichero DAM, se le asigna 'DIRECT'. En otros casos 'UNKNOWN'.
	FORM = variable *carácter*	Si el fichero está conectado para una E/S con formato, se le asigna 'FORMATTED' En caso contrario, se le asigna 'UNFORMATTED'. En otros casos, 'UNKNOWN'.	Si la unidad está conectada al fichero para una E/S con formato, se le asigna 'FORMATTED'. En caso contrario, se le asigna 'UNFORMATTED'. En otros casos, 'UNKNOWN'.

Cuadro 3.3

	ESPECIFICADOR	RESULTADO ASIGNADO A LA VARIABLE AL PREGUNTAR	
		POR FICHERO.	POR UNIDAD
SI EL FICHERO SE CONECTA PARA UN ACCESO DIRECTO	RECL = variable entera	Se le asigna el valor correspondiente a la longitud del registro, cuando el fichero está conectado para ser accedido mediante un acceso directo.	Se le asigna el valor correspondiente a la longitud del registro, cuando la unidad está conectada a un fichero DAM.
	NEXTREC = variable entera	Se le asigna el valor $n + 1$, siendo n el número del registro tratado en la última operación de E/S. Si no ha ocurrido ninguna operación de E/S desde que se haya abierto el fichero, n es *cero*.	Se le asigna el valor $n + 1$, siendo n el número del registro tratado en la última operación de E/S sobre el fichero conectado a esa unidad. Si no ha ocurrido ninguna operación de E/S desde que la unidad fue conectada al fichero, n es *cero*.

Cuadro 3.4

	ESPECIFICADOR	RESULTADO ASIGNADO A LA VARIABLE AL PREGUNTAR	
		POR FICHERO	POR UNIDAD
SI EL FICHERO SE CONECTA PARA UNA E/S CON FORMATO	BLANK = variable *carácter*	Se le asigna el valor 'NULL' cuando los campos aritméticos en una operación de entrada, se tratan como blancos. Se le asigna el valor 'ZERO', cuando esos campos se tratan como ceros.	Idem que el anterior *por unidad*, pero en lugar de considerar el fichero, tener en cuenta más bien el fichero conectado a esta unidad.

Cuadro 3.5

ERR=n e IOSTAT=v, con el mismo significado que el descrito en sentencias anteriores de E/S.

Como idea general, diremos que para un fichero de acceso secuencial o directo pueden preguntarse, utilizando esta sentencia, características como: existe o no el fichero, está o no conectado a una unidad, el número de su unidad, su nombre, su método de acceso o si tiene formato o no. En el caso de un fichero de acceso directo, podría preguntarse además por la longitud de sus registros o bien por el número del registro siguiente a tratar.

Los especificadores contenidos en la sentencia INQUIRE pueden darse en cualquier orden, no pudiendo aparecer más de una vez. Una misma variable tipo entera, *carácter* o lógica no puede pertenecer a más de un especificador.

Ejemplos:

a)
```
INQUIRE (FILE = A32, ERR = 300, NAMED = XLOG,
         SEQUENTIAL = SEC, NUMBER = IJ)
```

Se trata de una pregunta por nombre de fichero. Ese nombre es A32; si el fichero efectivamente tiene nombre, el contenido de la variable XLOG, que habrá sido definida como lógica, to-

mará, el valor ·TRUE·; otra variable definida como lógica es SEC e IJ es una variable entera, donde estará contenido el valor de la unidad externa conectada al fichero.

b) INQUIRE (UNIT = IN + 2, NUMBER = M2, ACCESS = ACCESO, RECL = NREG)

Se trata de una pregunta por número de unidad: la correspondiente a IN + 2.

En M2 estará contenido el número de la unidad conectada al fichero y si el acceso es directo, ACCESS contendrá 'DIRECT' y NREG la longitud del registro.

EJERCICIOS RESUELTOS

3.1. Indicar los errores, si existen, en cada una de las sentencias siguientes:

a) READ, *, A, B, C, D
b) READ *, 100, A, B, I, J
c) READ *, A(3), A(1), B
d) READ '(F5.3, F4.2)', A, II
e) READ (3F6.2, 'I3), A, B, C, JOTA
f) READ '(I, F5.1)', K, F

Respuesta

a) No debe escribirse coma (,) entre READ y *
b) Si 100 es información de formato no puede darse además * para indicar formato libre. Si 100 fuera un número de unidad, debería escribirse READ (100, *) A, B, I, J
c) Es correcta.
d) La variable II precisa código de formato entero I, mientras no se defina como real.
e) La declaración de formato es una variable *carácter* que se escribe entre paréntesis. Faltan las comillas: '(3F6.2, I3)' es la forma correcta.
f) En la especificación I falta el número de caracteres numéricos que contendrá la variable K.

3.2. Mediante un terminal de pantalla-teclado se introducen los datos siguientes que son leídos utilizando en el programa las sentencias READ que se indican.

Decir cuáles serán los valores que cada variable toma en memoria.

a) 3.25 __–38__453.83__24

READ *, X, MAT, Y, KILO

b) 43, 21, 83, 23.06, – 36.5
READ 100, J(1), J(3), J(5), A, B

Respuesta

a) X = 3.25
MAT = –38
Y = 453.83
KILO = 24

b) J(1) = 43
J(3) = 21
J(5) = 83
A = 23.06
B = 36.5

3.3. Analizar las sentencias de E/S siguientes y decir qué formato corresponde a cada variable de la lista.

a) READ '(3I4, F5.1)', I, J, K, A
b) READ '(A, I3, F5.1)', NOM, J, VEL(3)
c) PRINT '(I2, F4.1, A), NUM, AMA(2), TITUL

Respuesta

a) Para I, I4 (entera)
Para J, I4 (entera)
Para K, I4 (entera)
Para A, F5.1 (real)

b) Para NOM, A (constante *carácter*)
Para J, I3 (entera)
Para VEL(3), F5.1 (real)

c) Para NUM, I2 (entera)
Para AMA(2), F4.1 (real)
Para TITUL, A (constante *carácter*)

3.4. Suponiendo que el formato libre imprime las cantidades enteras en un máximo de seis posiciones y las reales en diez, cuáles serán los registros de salida producidos por las sentencias siguientes (indicar los blancos con _):

a) PRINT *, 'RADIO = ', 321, 'LONG = ', 1433.82
b) PRINT *, 'PRECIO = ', 1537, ' _ TOTAL = ', 83215.85
c) PRINT '(A, I4, A, F7.2)', 'BASE =', 50, ' _ AREA = ', 4621.5 * 2

Respuesta

a) RADIO = ___ 321LONG = ___ 1433.82 (Los datos se ajustan a la derecha)
b) PRECIO = __ 1537 _ TOTAL = __ 83215.85 (Idem que la anterior)
c) BASE = __ 50 _ AREA = 9243.00 (Se ajustan a la derecha dentro de las especificaciones I o F)

3.5. Siguiendo las instrucciones de los programas siguientes, determinar cuál será la salida impresa en cada caso.

a)
```
I = 2
J = 2
K = 3 * I + J
I = 2 * I
J = I + K + J
K = 2 * I + 5
PRINT *, I, J, K
```

b)
```
      I = 1
      J = 3
10    PRINT * , I
20    I = I + 1
      IF (I – J) 10, 30, 30
30    J = J + 3
      IF (I – 20) 20, 40, 40
40    STOP
      END
```

Respuesta

a) Se imprimen los valores siguientes: I = 4, J = 14 y K = 13 en una única línea.
b) Se imprimen distintos valores de la variable I, cada uno en una línea distinta.
Esos valores son: 1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19, 20

3.6. Dadas las instrucciones emparejadas siguientes, contestar las preguntas que se indican.

a)
```
DIMENSION X(2, 3), Y(5), V(10)
READ (1, 1050) U, V(3), X, Y
```

b)
```
DIMENSION X(8), Y(2,2,3), V(3)
READ (1, 100) U, V, X(7), Y
```

¿Cuántas variables se leen en total? ¿Cuántas variables del conjunto V se leen? ¿Y del X? ¿Y del Y?

Respuesta

a) Trece. Una: la V(3). Seis: todo el conjunto bidimensional. Cinco: todo el conjunto lineal.
b) Diez y siete. Tres: todo el conjunto. Una: la X(7). Doce: todo el conjunto.

3.7. Comentar los errores, si existen, de las sentencias siguientes, en las que 1 indica el terminal pantalla-teclado y 2 la impresora.

a) READ (200, 1, END = 1.3) A, B, C
b) WRITE (UNIT = 2, ERR = 30, END = 20, FMT = 100), A, B, C
c) READ (1,300, 33, 80) A, B, C

Respuesta

a) Si en la información de control no se utilizan palabras clave para unidad y formato, aquella debe preceder a éste. A END = tiene que seguirle una constante entera.
b) Sobra la coma después del paréntesis de cierre.
c) Faltan palabras clave en la información de control: END = y ERR =

3.8. Comentar los errores, si existen, de las sentencias siguientes:

a) READ(4) A(3), B(2,5), I
b) READ 1, A, C, I, MATRIZ
c) READ, A(I), B(2)
d) READ (IOSTAT = II, UNIT = 4)

Respuesta

a) y d) Son válidas.
b) El número de unidad, 1, tiene que estar encerrado entre paréntesis.
c) Le falta el número de unidad.

3.9. Encontrar los errores, si existen, en las siguientes sentencias WRITE y PRINT:

a) WRITE (*, 100) A(5), B, MUL
b) WRITE (200, 2)
c) PRINT FMT = 200, A, B, C
d) WRITE (300, UNIT = 2) D, E, F
e) WRITE (IN + 4, NOM, ERR = 999) C(5), B(2,3)
f) WRITE (FMT = 200, 4) I, J, K(3)

Respuesta

a) y e) Son correctas.
b) El número de unidad ,2, debe colocarse delante del formato 200.
c) Las sentencias PRINT no contienen la palabra FMT =
d) Tiene que utilizarse la opción FMT = para el formato 300 ya que figura UNIT =
f) El número de unidad ,4, debe colocarse delante del formato FMT =

3.10. Suponiendo las siguientes sentencias READ y WRITE de acceso secuencial y sin formato, analizar cuáles son correctas y cuáles no.

a) WRITE 6, A, C(3), D
b) WRITE (UNIT = 6) X, Y, Z
c) READ (6)
d) READ, A, B, C
e) READ 6, C, D, E

Respuesta

a) La unidad 6 tiene que estar entre paréntesis.
b) y c) Son correctas.
d) Tiene que indicarse la unidad lógica.
e) El número de unidad debe ir encerrado entre paréntesis.

3.11. Encontrar los errores, si existen, en las siguientes sentencias auxiliares de manipulación de cintas magnéticas.

a) ENDFILE UNIT = 6
b) ENDFILE (ERR = 100, IOSTAT = 5, 6)
c) REWIND (2 * IN – 5, END = 23, ERR = 100)
d) REWIND (UNIT = 3 * J + 4)
e) BACKSPACE (IOSTAT = IO, ERR = 100, UNIT = 3 * J –5)
f) BACKSPACE 6, ERR = 100

Respuesta

a) UNIT = 6 debe estar encerrado entre paréntesis.
b) Tiene que escribirse UNIT = 6 o bien escribir el número de la unidad, 6, al principio de la lista de opciones. Después de IOSTAT = debe seguir una variable entera.
c) Esta correcta. El número de la unidad viene dado por la expresión 2 * IN –5 que es entera.
d) Correcta.
e) Correcta.
f) Tiene que emplearse paréntesis para encerrar la lista de opciones.

3.12. Las siguientes sentencias corresponden a E/S de acceso directo con formato. Comentar los errores, si existen.

a) READ (3,200, ERR = 10, IOSTAT = J) A, B
b) READ (REC = 30, UNIT = 3, FMT = 100) C, D, E
c) WRITE (300,3) X, Y
d) WRITE (300, UNIT = 3, ERR = 50) K, J, L
e) READ (3, REC = 25) X, A, C(2)

Respuesta

a) Falta la indicación del registro a partir del cual se lee, REC =
b) Es correcta.
c) La unidad lógica 3 tiene que preceder al número de formato 300. Falta REC =
d) Falta REC = necesario en el acceso directo.
e) Falta la indicación de formato.

3.13. Encontrar los errores, si existen, en las siguientes sentencias de acceso directo sin formato.

a) READ (3, 200, REC = 30) A, B, C, D
b) READ (UNIT = 3) C, D
c) WRITE (REC = 10, UNIT = 3, ERR = 5) C, E, J
d) WRITE (REC = 25, 3, ERR = 100) Z, W

Respuesta

a) No tiene que darse la indicación de formato 200.
b) Tiene que indicarse la opción REC = en el caso directo.
c) Es correcta.
d) Como el número de la unidad lógica no figura en primer lugar en la lista de opciones, tiene que escribirse UNIT = 3.

3.14 Dadas las sentencias siguientes, escribirlas correctamente.

a) READ (M, 200) (A(I), I = 1,4), B, (C(I), I = 1, 9, 2)
b) WRITE (N, 300), (A, B, (C(J), J = 1, M, 2))
c) READ (N, 100), A, (B(K) K = 1,20)

Respuesta

a) Es correcta.
b) WRITE (N,300) A, B, (C(J), J = 1,M,2)
c) READ (N,100) A, (B(K), K = 1,20)

3.15 Dada la sentencia

WRITE (L,10) A(3), B, (X(I), Y(I), I = 2, 7, 2)

escribir en su orden las variables producidas en la salida.

Respuesta

A(3), B, X(2), Y(2), X(4), Y(4), X(6), Y(6)

3.16. Dada la sentencia

WRITE (N, 300) A, H(4), ((X (M, N), M = 2,3), N = 1,3)

¿cuál será el conjunto ordenado de variables escritas?

Respuesta

A, H(4), X(2,1), X(3,1), X(2,2), X(3,2), X(2,3), X(3,3)

3.17. Enumerar las variables leídas empleando la sentencia

```
READ (N,100)  (X(I), I = 1,3),  ((A(J, K), K = 1,2), J = 1,3)
```

Respuesta

X(1), X(2), X(3), A(1,1), A(1,2), A(2,1), A(2,2), A(3,1), A(3,2)

3.18 Enumerar en su orden las variables escritas de acuerdo con la sentencia siguiente:

```
WRITE (N, 200) ((X (I, J), I = 2,8,3), J = 1,4)
```

Respuesta

X(2,1), X(5,1), X(8,1), X(2,2), X(5,2), X(8,2), X(2,3), X(5,3), X(8,3), X(2,4), X(5,4) y X(8,4)

3.19 Enumerar las variables leídas mediante la sentencia

```
READ (UNIT = 4) X, Y, (Z(I), I = 1,3), W(5)
```

Respuesta

X, Y, Z(1), Z(2), Z(3) y W(5)

3.20. Dada la codificación

```
DIMENSION X(3,3)
READ (M, 400)  A,  ((X(I, J), I = 2,3), J = 1,3)
```

construir un esquema como el explicado en 3.12 donde se refleje el orden de entrada de las variables, las variables y la ubicación que les corresponde dentro de la memoria.

¿Qué sucede con las ubicaciones 1, 4 y 7?

Supóngase que en el programa no existen más variables que las A y las X.

Respuesta

ORDEN DE ENTRADA		2º	3º		4º	5º		6º	7º	1º
VARIABLE		X(2,1)	X(3,1)		X(2,2)	X(3,2)		X(2,3)	X(3,3)	A
UBICACION	1	2	3	4	5	6	7	8	9	10

ya que la sentencia DIMENSION reserva memoria en primer lugar para el conjunto X al principio de la codificación propuesta.

Las ubicaciones 1, 4 y 7 quedan vacías, ya que la variable I comienza su variación en 2 (I = 2,3)

3.21. Analizar las siguientes sentencias y deducir los errores, si existen.

a) OPEN (UNIT = 5, FILE = 'AMARO', ACCESS = 'SEQUENTIAL', RECL = 80)
b) OPEN (N, FILE = 'XXX', ERR = 100, FORM = 'UNFORMATTED', BLANCK = 'NULL')
c) CLOSE (M+N, ERR = 200, IOSTAT = A, STATUS = 'KEEP', FILE = 'BB')
d) Definido el fichero A324 como fichero de trabajo, se cierra empleando la instrucción

CLOSE (5, ERR = 300, IOSTAT = MAX, STATUS = 'KEEP')

Respuesta

a) Tratándose de un fichero secuencial, la opción RECL= no debe figurar.
b) Con un fichero sin formato no puede emplearse la opción BLANK =
c) En la instrucción CLOSE no figura la opción FILE =. La variable después de la opción IOSTAT = tiene que ser entera.
d) Un fichero de trabajo (definido como SCRATCH) no puede guardarse, es decir, no puede emplearse la opción STATUS = 'KEEP'

3.22. Se trata de escribir una sentencia OPEN que contenga los parámetros u opciones mínimas correspondientes a las siguientes definiciones:

a) Fichero SAM ya existente, catalogado con el nombre FICHE sobre la unidad IN.
b) Idem que el anterior pero DAM y con formato.
c) Idem que a) pero con control del error.
d) Idem que b) pero el fichero es ahora nuevo.
e) Idem que a) pero el fichero es ahora de trabajo.

Respuesta

a) OPEN (UNIT = IN, FILE = 'FICHE', ACCESS = 'SEQUENTIAL', STATUS = 'OLD')
b) OPEN (IN, FILE = 'FICHE', ACCESS = 'DIRECT', STATUS = 'OLD', FORM = 'FORMATTED')

c) OPEN (IN, FILE = 'FICHE', ACCESS = 'SEQUENTIAL', STATUS = 'OLD', IOSTAT = II)

d) OPEN (IN, FILE = 'FICHE', ACCESS = 'DIRECT', STATUS = 'NEW', FORM = 'FORMATTED')

e) OPEN (FILE = 'FICHE', ACCESS = 'SEQUENTIAL', STATUS = 'SCRATCH', UNIT = N)

4

Códigos

Lo superfluo es algo muy necesario

VOLTAIRE *(1694-1778)*

4.1. SENTENCIA Y CODIGOS DE FORMATO

En el capítulo anterior ya nos hemos referido a algunos códigos de formato utilizados al tratar las sentencias de E/S. El presente capítulo estará dedicado a ampliar esta información: la relativa a la sentencia de formato y los códigos utilizados en la misma.

Ya sabemos que la información, en la memoria del ordenador, se almacena de una forma específica a base de ceros y unos, es decir, en binario; también sabemos que esa representación interna no coincide con la externa, es decir, con la que podemos ver a través de una pantalla del terminal o bien la impresa en un documento que normalmente viene dada en forma decimal. Existe, por consiguiente, una conversión entre ambas representaciones. Este es uno de los cometidos que tienen las sentencias de formato.

Generalizando: es necesario que en todo momento el sistema de cálculo u ordenador que se emplee, tenga la suficiente información relativa a la *clase, tamaño* y *forma de los datos* con los que va a trabajar. Para ello se emplean sentencias de formato, FORMAT, las cuales acompañan a cada una de las sentencias de E/S que existan en un programa.

Examinemos un poco más de cerca cada uno de los tres aspectos anteriores.

Cuando hablamos de información relativa al *tamaño* de los datos, nos referimos al número de posiciones que cada uno de ellos ocupa, al ser perforado en una ficha, al ser tecleado en un terminal, al ser impreso en una impresora, por ejemplo. Así decimos que la parte entera de un número ocupa cuatro posiciones en una línea de una pantalla y la fraccionaria dos.

En el capítulo primero ya hablamos de datos enteros, reales, *carácter*, etc. Pues bien; las sentencias de formato facilitan al sistema la información suficiente para que éste pueda reconocer cuando un dato es entero, lógico, doble precisión, etc., es decir, su *clase*. Esto se consigue empleando en la sentencia de formato o FORMAT unos *códigos* apropiados.

Recordando el párrafo 1.4 en lo relativo a como estaba constituida la memoria principal de un ordenador, vamos ahora a ver un ejemplo de como se almacenan los datos FORTRAN en una máquina. Existen varias formas dependiendo del sistema que se utilice, pero una de las mas frecuentes, sobre todo cuando se trata de sistemas grandes, es la siguiente:

a) *Datos enteros:* se almacenan en forma binaria, ocupando una palabra de 32 bits, con la distribución que se indica en la figura 4.1.

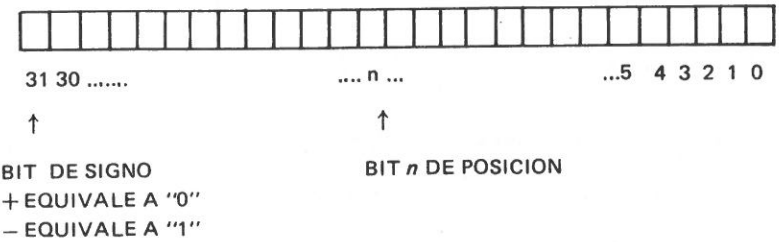


Figura 4.1

Así un 1 en la posición 0 significa, $2^0 = 1$; un 1 en la posición 5, significa $2^5 = 32$; un 1 en la posición n, significa 2^n. Por ejemplo, la constante entera 564, expresada en notación habitual decimal, equivale a una notación binaria interna constituida por la configuración siguiente:

$$00000000000000000000001000110100 = 2^2 + 2^4 + 2^5 + 2^9$$

b) *Datos reales:* en este caso la configuración interna está formada por dos partes: una para la mantisa y otra para el exponente, de manera que la cantidad está siempre normalizada, es decir, considerando que el punto decimal precede al primer dígito significativo de la mantisa. Un esquema típico se indica en la figura 4.2 y ocupa 32 bits.

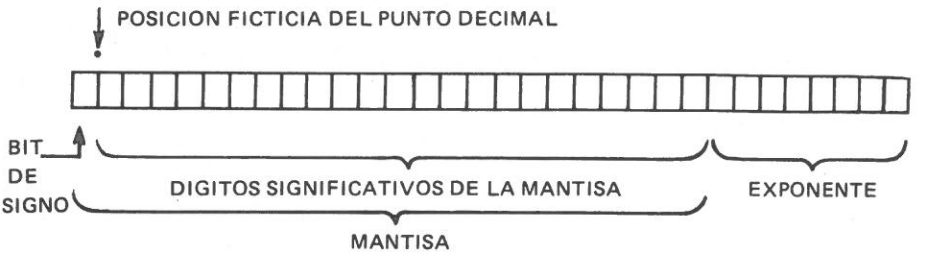


Figura 4.2

Teniendo en cuenta las siguientes equivalencias entre la representación decimal y binaria de un número real tal como el 22.375, por ejemplo, se verifica que

$$\underbrace{0.22375 \times 10^{2} = 22.375}_{\text{REPRESENTACION DECIMAL}} \equiv \underbrace{10110.011 = 0.10110011 \times 2^{101}}_{\text{REPRESENTACION BINARIA}}$$

Con ello, la representación interna del número positivo real 22.375, sería:

$$\underbrace{01011001100000000000000}_{\text{MANTISA CON SIGNO}} + \underbrace{00000101}_{\text{EXPONENTE}}$$

Podemos resumir diciendo que: las constantes decimales externas enteras se transforman en binarias internas, también enteras; las externas reales, se almacenan en binario pero siempre en forma normalizada, es decir, precediendo el punto decimal a las cifras significativas de la mantisa.

Pues bien; la sentencia FORMAT, facilita al sistema de cálculo la información necesaria para que éste reconozca la *forma* que los datos van a tener dentro de la memoria principal, enteros o en forma flotante normalizada, o bien la que van a tener al salir al exterior, enteros, reales, sin exponente o con exponente, etc., realizándose la conversión decimal-binario o viceversa, correspondiente. Por ello y ya en el capítulo anterior, se dijo que cuando la información no sufre conversión, binario-binario, no se utiliza la sentencia de formato.

Las sentencias de formato no son ejecutables sino únicamente informativas y pueden colocarse en cualquier lugar del programa fuente, aunque es aconsejable que todas ellas estén reunidas bien al principio o al final del programa.

La forma general de una sentencia de formato posee la sintaxis siguiente:

```
n FORMAT (lista de códigos de formato)
```

siendo: *n* una constante entera que indica el número o etiqueta de la sentencia **FORMAT**, que debe escribirse, por consiguiente, entre las columnas 1 a 5.

lista de códigos de formato, indica un conjunto de códigos especiales, separados por comas, y que sirven para describir y especificar clase, tamaño y forma de conversión de los datos, así como otras características especiales.

Los códigos de formato pertenecen a tres categorías distintas:

a) *Códigos para datos:* estás asociados con los datos numéricos o alfanuméricos que figuran en las sentencias de E/S. Son los siguientes:

Código	Tipo de dato
I	Entero
F	Real sin exponente
E	Real con exponente
D	Doble precisión
G	Real general
L	Lógico
A	Carácter

b) *Códigos de posicionamiento.*

Código	Propósito
T TL TR X	Tabulaciones
/	Comenzar un nuevo registro físico

c) *Códigos especiales:* utilizados para objetivos particulares. Se estudiarán más adelante y son los siguientes: SP, SS, S, P, BN, BZ, H, (') y (:).

Veamos ahora algunos ejemplos de la sentencia FORMAT.

a) 30 FORMAT (I5, F8.4)

El número de esta sentencia es 30; esta sentencia como todas las FORMAT vendrá asociada con otra de E/S y desde la cual estará referenciada. Posee dos códigos de formato: uno, el I, para datos enteros y otro, el F, para datos reales. Se estudiará el significado de los elementos siguientes a los códigos.

b) 80 FORMAT (F8.2/E10.4)

Sentencia de etiqueta 80, con dos códigos, F y E para números reales y que tratará dos registros (/).

c) 100 FORMAT (I4, F5.1 / E10.4, D8.2)

Sentencia número 100, para tratar cuatro datos distribuidos en dos registros físicos diferentes con los códigos respectivos I, F, y E, D.

4.2. CODIGO I

Este código (del inglés Integer) se utiliza en la sentencia FORMAT, cuando los datos que van a transferirse desde o a la memoria son de tipo entero.

La forma general de este código (más adelante estudiaremos otra más amplia) es:

siendo I el carácter de control o código propiamente dicho para tratar datos enteros y *n* una constante entera, sin signo, y distinta de cero, que indica la longitud del dato numérico, es decir, el número máximo de dígitos que posee. En esa longitud debe tenerse en cuenta la posición ocupada por el signo, cuando exista.

El código puede emplearse en la entrada o salida de cantidades.

a) *Entrada:* En este caso se transfieren desde el soporte externo a la memoria *n* caracteres, entre dígitos, blancos y signo (+ ó −). Si el dato está formado únicamente por blancos (se indicarán por _ en los ejemplos) se interpretarán todos ellos como que representan el valor *cero*.

Ejemplos:

VALOR EXTERNO	CODIGO	VALOR EN MEMORIA
8235	I4	8235
_−67	I4	−67
−337	I4	−337
453	I3	453
_____	I5	0
+453	I4	453

Obsérvese que entre las posiciones a contabilizar en *n* ha de tenerse en cuenta la ocupada por el signo, cuando este venga representado.

Cuando existan blancos entre los dígitos de un dato entero o bien al final del mismo, normalmente (dependiendo del sistema) se ignoran y los dígitos se ajustan a la derecha, al menos que al abrir el fichero con OPEN, se utilice la opción BLANK = 'ZERO' de la que ya se ha hablado. (Véase opción 8 de 3.13).

Veamos algunos otros ejemplos:

a) Las instrucciones

```
      READ (1, 500) IMA, ISO, JL, KM
500   FORMAT (I3, I4, I4, I1)
```

realizaría la siguiente asociación de variables y códigos:

```
IMA → I3
ISO → I4
JL  → I4
KM  → I1
```

b) Las dos sentencias

```
      READ (1,100) KIL1, KIL2, KIL3
100   FORMAT (I3, I2, I2)
```

servirían para introducir en memoria, a través de un terminal, la información

```
_–2304783 + 28
```

tecleada a partir de la primera columna y realizando la siguiente asociación:

```
KIL1 =  –2
KIL2 =  30
KIL3 =  47
```

El resto de la información contenida en la línea teclada, se ignoraría.

En la sentencia FORMAT del segundo ejemplo, podría hacerse una simplificación:

```
100   FORMAT  (I3, 2I2)
```

es decir, el código I2 repetido dos veces, puede sustituirse por 2I2.

c) La información

```
4372–23 ____1378
```

con

```
      READ (1, 33) KILO, JOTA, LAMA, MUN
33    FORMAT (I4, 2I3, I5)
```

ingresaría en memoria así:

```
KILO  → IA → 4372
JOTA  → I3 →  –23
LAMA  → I3 →    0
MUN   → I5 → 1378
```

b) *Salida:* Se transfieren desde memoria el soporte externo, *n* caracteres. Si el número de dígitos significativos que forman el dato

internamente junto con el signo ocupan una longitud menor que n, las posiciones de la izquierda de la salida se rellenan con blancos; si totalizan un número mayor que n, la salida está formada por n asteriscos (*) en lugar del propio dato.

Si el valor es negativo, se coloca un signo menos (–) delante del primer dígito significativo de la izquierda.

Ejemplos:

a) Los siguientes valores contenidos en memoria saldrán al exterior de la forma siguiente:

VALOR INTERNO	CODIGO	VALOR EXTERNO
824	I3	824
0000	I3	_ _0
-5	I4	_ _ _-5
1723	I3	***
-34	I5	_ _ _-34

b) Supónganse las sentencias

```
      WRITE (1, 200) I, J, K
200   FORMAT (I4, I3, I5)
```

Si los valores contenidos en memoria fueran

```
I =  –315
J =    23
K =  –217
```

la salida por la pantalla del terminal, en una misma línea, sería:

```
–315_23_–217
```

con lo cual, cada dato, estaría separado del siguiente por un carácter blanco.

c) Sean los siguientes valores en memoria:

```
KMS   = 1483
LUAS  = –475
```

```
MAS   =    23
MEN   =17450
```

empleando la codificación

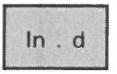
```
      WRITE (2,30) KMS, LUAS, MAS, MEN
30    FORMAT (I3, 2I6, I7)
```

se produciría, por una impresora, la línea siguiente:

```
***__-475____23__17450
```

lo cual indica que la especificación I3 no es suficiente para imprimir el dato de cuatro dígitos 1483.

Otra forma que puede adoptar el código I es (no existe en el FORTRAN básico).



siendo *d* una constante entera sin signo que indica el número mínimo de dígitos que aparecerán en el registro de salida. En la entrada, un código In.d equivale al In.

Ejemplos:

a) El dato interno 125, al emplear el código I10.5, saldría como

```
_____00125
```

es decir, una longitud de 10 y al menos 5 dígitos. Cuando no existe ese total de dígitos significativos, se rellena el campo con ceros a la izquierda.

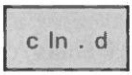
b) El dato interno 0 (cero), con el código I3.0, daría lugar a

```
___
```

c) El dato −1724 con I10.7, saldría como

```
__-0001724
```

Con todo lo dicho, la forma más general del código I es

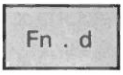


siendo *c* un coeficiente (optativo) de repetición que es una constante entera mayor que cero y sin signo.

4.3 CODIGO F

El código F (del Inglés Float) se utiliza para transferir datos reales básicos (véase 1.6, a), es decir, valores que externamente tienen una parte entera y una parte fraccionaria, separadas ambas por un punto decimal y un signo opcional precediendo a todos los dígitos.

Su forma general es:



siendo F el código, *n* una constante entera, sin signo, no nula, denominada *anchura del campo*, que representa el total de posiciones ocupadas por la cantidad que va a leerse o escribirse; *d* es otra constante entera sin signo, que indica el total de cifras decimales que posee la parte fraccionaria del dato. En *n* debe contabilizarse el signo, si existe, y el punto decimal además de los propios dígitos.

Como ya hicimos con el código I, estudiaremos el F considerando separadamente las entradas de las salidas. Debemos resaltar que pueden existir algunas diferencias en cuanto a cómo cada compilador trata el código F. Una vez más aconsejamos leer los manuales de la propia instalación de trabajo.

a) *Entrada:* Se transfieren a la memoria *n* caracteres entre los cuales debe incluirse el signo, si existe, los dígitos propiamente dichos y el punto decimal, si viene explícitamente indicado. Si el punto decimal no figura (no se ha tecleado, no se ha perforado) entonces el sistema considera automáticamente como parte fraccionaría del dato, los *d* dígitos de más a la derecha.

Además, el dato real básico puede estar acompañado (seguido) por un exponente que puede adoptar cualquiera de las tres formas siguientes:

- Una constante entera precedida de un signo + (más) o – (menos)
- Una **E** o una **D** seguidas de un signo opcional y de una constante entera. Para el significado de **E** y **D**, véanse 1.6, b) c) y 1.9, a), b), respectivamente.

Ejemplos

VALOR EXTERNO	CODIGO	VALOR DE MEMORIA
49.225	F6.3	$49.225 \equiv 0.49225 \times 10^2$
49.225	F6.2	$49.225 \equiv 0.49225 \times 10^2$
–1525.1	F7.1	$-1525.1 \equiv -0.15251 \times 10^4$
–35232	F6.4	$-3.5232 \equiv -0.35232 \times 10^1$
433.05	F6.0	$433.05 \equiv 0.43305 \times 10^3$
43305	F5.2	$433.05 \equiv 0.43305 \times 10^3$
43305E –2	F8.0	$433.05 \equiv 0.43305 \times 10^3$
43.305 D1	F8.3	$433.05 \equiv 0.43305 \times 10^3$
43305 –2	F7.0	$433.05 \equiv 0.43305 \times 10^3$

En estos ejemplos debe observarse lo siguiente: en memoria se ha representado también el valor real como estará almacenado, es decir, en forma de mantisa normalizada con exponente (en sistema decimal); los dos primeros códigos producen el mismo resultado, lo cual indica que cuando el punto decimal figura explícitamente en el dato, la expecificación *d* es indiferente (véanse también los ejemplos quinto y sexto).

Otros ejemplos:

a) Supóngase que la información contenida en las veinte primeras columnas de una línea de terminal fuera

```
_13834–61532__837419
```

y que esta se leyera mediante las sentencias:

```
      READ (1, 100) A, B, C
100   FORMAT (F6.2, F6.1, F8.4)
```

Entonces los valores respectivos asignados en memoria a las variables, serían:

```
A =  138.34
B =  –6153.2
C =  83.7419
```

b) La información perforada en una ficha es

```
–3425D–4__835__44.21325E+2
```

y se lee en memoria mediante las sentencias

```
      READ (4, 300) A, KM, C, X
300   FORMAT (F8.0, I5, F6.4, F7.0)
```

Entonces los valores asignados en memoria serían:

```
A = –0.3425
KM = 835
C =  44.2
X =  132500.0
```

b) *Salida:* Todos los caracteres transferidos desde la memoria al soporte externo, se colocan ajustados a la derecha de un campo de anchura n. Por ello la salida consta de blancos, si fueran necesarios, seguidos del signo menos si el dato interno es negativo y de los dígitos del dato con su punto decimal.

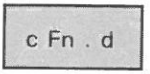
No se producen ceros a la izquierda del número, excepto si éste es completamente fraccionario (menor que uno), en cuyo caso, existirá un cero a la izquierda del punto decimal. En algunos sistemas este cero no existe y el dato comienza directamente por ese punto decimal.

Si la parte fraccionaria del dato tiene internamente más anchura que la reservada en d, su valor se redondea antes de ajustarse a d posiciones. Si el dato completo no cabe en n posiciones, se produce una salida de n asteriscos, indicando error.

Ejemplos:

VALOR INTERNO	CODIGO	VALOR EXTERNO
$0.25338 \times 10^{2} \equiv 25.338$	F9.3	___25.338
$0.35247 \times 10^{0} \equiv 0.35247$	F9.1	_____0.4
$0.89235 \times 10^{-1} \equiv 0.089235$	F6.2	__0.09
$-0.73256 \times 10^{3} \equiv -732.56$	F7.1	_-732.6
$-0.12345 \times 10^{2} \equiv -12.345$	F4.3	****
$-0.12345 \times 10^{2} \equiv -12.345$	F6.2	-12.35
$0.12468 \times 10^{3} \equiv 124.68$	F10.2	____124.68
$0.35000 \times 10^{0} \equiv 0.35000$	F4.2	0.35

El código F puede repetirse escribiendo delante del mismo un coeficiente *c* constante entera, positiva, no nula. La forma general es entonces:



Otros ejemplos:

a) Las variables A(1), A(2), A(3) cuyos valores respectivos son

```
 7.3341
-83.70
-124.351
```

se teclean en un terminal a partir de la columna número uno así:

```
73341-8370-124351
```

Entonces, las sentencias correctas para leerlas, serían:

```
      READ (1,100) A(1), A(2), A(3)
100   FORMAT (F5.4, F5.2, F7.3)
```

b) La sentencia

```
FORMAT (I3, I3, F3.1, F4.2, F4.2, F4.2)
```

simplificadamente se escribiría así:

```
FORMAT (2I3, F3.1, 3F4.2)
```

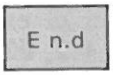
4.4. CODIGOS E, D y G

El código E (del Inglés, Exponent) se emplea para manejar datos reales con exponente, es decir, datos que en las salidas aparecerán acompañados del exponente E y una característica (forma de coma flotante); en cuanto a la entrada, este código es equivalente al F que tuviera la misma anchura de campo.

El código D (del Inglés, Double Precision) se utiliza para manejar datos declarados como de doble precisión (Véanse 1.9 y 1.13). Su uso práctico es equivalente al de E. Unicamente, debemos advertir que, en las salidas, las cantidades en coma flotante están acompañadas de un exponente D en lugar de E. Este código no puede utilizarse en el FORTRAN básico ya que en él no existen los datos en doble precisión.

El código G (del Inglés, General) tiene una misión parecida a los F y E. Será estudiado más adelante. No existe en el FORTRAN básico.

La forma general del código E es



siendo E el código, *n* una constante entera, sin signo, no nula que representa la *anchura del campo,* en la cual debe considerarse: el signo de la mantisa cuando exista, el dígito que precede al punto decimal, el punto decimal, la mantisa y el exponente; *d* es otra constante entera que indica el número de dígitos que tiene la mantisa.

a) *Entrada:* el código E realmente tiene el mismo funcionamiento que el F cuando se trata de la entrada de datos. No obstante haremos un nuevo análisis de este tema.

En este caso se transfieren desde el exterior a memoria *n* caracterés: el signo que es opcional, seguido por un dígito al menos, que precede al punto decimal que es también opcional. Si este punto no

figura, entonces se considera como parte fraccionaria los *d* dígitos de más a la derecha de la mantisa. A continuación vendrá la parte fraccionaria seguida del carácter E y del exponente. La E puede omitirse si el exponente viene precedido de los signos ±.

Ejemplos:

VALOR EXTERNO	CODIGO	VALOR EN MEMORIA
–43.258E3	E9.3	$-43.258 \times 10^{3} \equiv -0.43258 \times 10^{5}$
–43.258E3	E9.5	$-43.258 \times 10^{3} \equiv -0.43258 \times 10^{5}$
0.45258 –02	E11.5	$0.45258 \times 10^{-2} \equiv 0.45258 \times 10^{-2}$
42505E +3	E8.2	$425.05 \times 10^{3} \equiv 0.42505 \times 16^{6}$
–257659 –2	E9.1	$-25765.9 \times 10^{-2} \equiv -0.257659 \times 10^{3}$
–1.2567E25	E9.4	$-1.256 \times 10^{2} \equiv -0.12565 \times 10^{3}$
+493327 + 03	E10.3	$493.327 \times 10^{3} \equiv 0.493327 \times 10^{6}$
–1.253E2	E8.0	$-1.253 \times 10^{2} \equiv -0.1253 \times 10^{3}$

En estos ejemplos puede observarse que: en memoria se ha representado también el valor tal como realmente estará almacenado, es decir, en forma de mantisa normalizada con exponente (en el sistema decimal); los dos primeros códigos producen la misma entrada, es decir, cuando el punto decimal viene indicado explícitamente, la especificación *d* del código es indiferente; para el exponente, por ejemplo, son equivalentes las formas E + 02, E + 2, E2, + 02, + 2; en el caso sexto se ha dado una especificación de anchura del campo insuficiente, nueve en lugar de diez, y por ello el exponente ingresado ha sido 2 en lugar de 25; en el ejemplo séptimo figura explícitamente el signo más (+) de la mantisa y por ello se ha contabilizado en la anchura del campo.

Otros ejemplos:

a) Sean las variables

```
OMEGA = 34.837E02
BETA = 0.00077E-1
X = 721.42E–7
```

que se introducen por un terminal, tecleando

```
34837E02__00077E−1_72142E−7
```

Para que fueran correctamente ingresadas en memoria debería codificarse así:

```
      READ (1,100) OMEGA, BETA, X
100   FORMAT (E8.3, E10.5, E9.2)
```

b) Si los datos anteriores se teclearan como

```
34.837E02.00077E−01721.42E−07
```

entonces la sentencia FORMAT necesaria, sería ahora:

```
FORMAT  (E9.3, E10.5, E10.2)   o bien
FORMAT  (E9.0, E10.0, E10.0)
```

b) *Salida:* en este caso, se transfieren desde la memoria al soporte externo y en forma normalizada *n* caracteres en total y únicamente tantos decimales como indica *d*. En la anchura del campo *n* ha de reservarse una posición para el punto decimal, otra para el dígito *cero* que precederá a dicho punto, cuatro para el exponente y otra para el signo algebráico (−) de la mantisa cuando ésta sea negativa. Si la anchura del campo *n* es mayor que las posiciones necesarias para el resultado, el dato sale precedido de blancos; si *n* es menor se imprimen *n* asteriscos.

Si la mantisa contenida en memoria no sale completa, el dato obtenido como resultado se redondea.

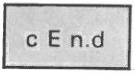
Obsérvese que *n* debe ser al menos 8 para poder acomodar el signo de la mantisa, el cero que precede al punto decimal, el propio punto decimal, al menos un dígito para la mantisa y cuatro caracteres para el exponente que es siempre de la forma

```
E ± xx
```

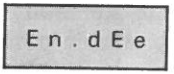
Ejemplos:

VALOR INTERNO	CODIGO	VALOR EXTERNO
0.83974×10^{2}	E10.2	__0.84E + 02
0.83974×10^{2}	E10.3	_0.840E + 02
0.83974×10^{2}	E10.4	0.8397E + 02
0.83974×10^{2}	E9.4	* * * * * * * * *
-0.63×10^{1}	E9.1	_–0.6E +01
-0.13579×10^{3}	E12.5	–0.13579E + 03
0.2468×10^{5}	E12.4	__0.2468E + 05
-0.3482×10^{-1}	E9.2	–0.35E –01

El código E puede repetirse en la sentencia FORMAT, en cuyo caso vendrá precedido de una constante entera c:



El código E puede reservar más de dos dígitos para el exponente a la salida de resultados. Esto se logra utilizando otra forma general:



Los valores de *n* y *d* ya son conocidos. En cuanto a *e*, ésta es otra constante entera positiva no nula que indica el número de dígitos que va a tener el exponente. En este caso, el exponente ocupará *e+2* dígitos en lugar de cuatro como se había establecido anteriormente.

En la entrada de datos, esta nueva forma no tiene efecto alguno, es decir, equivale a la estudiada en primer lugar.

Ejemplos:

a) 0.246810×10^{3} con E14.3E3 saldría como

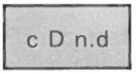
____ 0.247E+003

b) 0.13579×10^2 con E10.2E4 saldría como

0.14E–0002

Como ya hemos dicho, el código D se utiliza y tiene el mismo efecto que el código E. Su forma genera es

y si se usa repetido



Ejemplos:

a) El valor doble precisión en memoria

-0.43976743×10^9

saldría, utilizando el código D16.7, como

__–0.439764D+09

b) El valor externo

374.792D+3

ingresará en memoria correctamente utilizando, por ejemplo, el código D10.0 o bien el D10.3.

El código para tratar números reales en forma general ya hemos dicho que es G. Sus formas son

o

Los parámetros *n*, *d* y *e* ya han sido descritos anteriormente y tienen aquí el mismo significado que entonces. Veamos ahora su uso.

a) *Entrada:* tiene el mismo efecto que los códigos F o E.

b) *Salida:* si internamente el punto decimal se encuentra efectivamente situado entre dos dígitos cualesquiera de la mantisa (una vez desnormalizada de acuerdo con el exponente que posea), entonces el resultado que se obtiene es similar, aunque no igual, al que se produciría empleando el código F con estas dos diferencias:

- el campo contendrá a la salida cuatro blancos si se emplea la forma Gn.d o bien *e+2* blancos si se emplea la Gn.dEe en su parte de más a la derecha del resultado.
- con este código, *d* indica el número total de dígitos obtenidos en la salida y no sólo los correspondientes a la parte fraccionaria como ocurría con el empleo del código F.

En ambos casos, es decir, tanto con Gn.d como con Gn.dEe, el campo a la salida se rellena con blancos a la izquierda hasta completar la anchura *n*.

Por ejemplo, sea el dato en memoria

0.25338×10^2

que equivale al 25.338 con el punto decimal entre dos dígitos. Empleando el código F11.3 saldría como

_____25.338

con tres dígitos en su parte fraccionaria.

Empleando el código G11.5, el resultado obtenido ocupará *once* posiciones en total, con *cinco* dígitos efectivos, *cuatro* blancos a la derecha, el punto decimal y rellenándose con blancos a la izquierda hasta completar la anchura de *once* posiciones, es decir,

25.338____

Con el código G13.5E2 el resultado sería:

_25.338_____

Sea ahora el valor en memoria

$0.124620 \times 10^5 \equiv 12462.0$

al imprimirlo con el código G14.6 saldría como

```
___12462.0____
```

y al imprimirlo con G16.6E4 el resultado sería

```
___12462.0______
```

Igualmente, el dato en memoria 0.2468375 x 10^4 ≡ 2468.375 al emplear los códigos G13.6 y G15.7E4 saldrá, respectivamente, como

```
__2468.38____
_2468.375______
```

Si la localización efectiva del punto decimal no se encuentra entre los dígitos de la mantisa, entonces el código G equivale al E. Por ejemplo, el valor interno 0.85729 x 10^{-1} ≡ 0.085729 mediante el código G12.5 saldría como

```
_0.85729E-01
```

Igualmente, el dato interno 0.358476 x 10^6 ≡ 358476 saldrá con el código G13.6 como

```
_0.358476E+06
```

El mismo dato, empleando G14.6E4 saldría como

```
0.358476E+0004
```

El código G también puede repetirse dentro de la sentencia FORMAT utilizando un coeficiente *c* precediendo al propio carácter G.

4.5. CODIGO L

Este formato se emplea para manipular datos lógicos. (Véase 1.7). Su forma general es

donde L (del Inglés, Logical) es el código y *n* la anchura del campo, es decir, el número de posiciones que ocupa el dato lógico.

a) *Entrada:* el código da lugar a que se realice una búsqueda en los *n* primeros caracteres del campo, comenzando por la izquierda. Si el primer carácter es T o bien .T, la constante a ingresar será .TRUE. . Si el primer carácter encontrado es F o bien .F, el dato a ingresar será .FALSE. . Los demás caracteres del campo, hasta completar los *n*, se ignoran.

Ejemplos:

VALOR EXTERNO	CODIGO	VALOR EN MEMORIA
·TRUE·___	L9	.TRUE··
__·T__	L6	·TRUE·
_·FAL__	L7	·FALSE·
_·TRAUS__	L9	·TRUE·
_F_	L3	·FALSE·
T__	L3	·TRUE·

b) *Salida:* se escribe la letra T o la F en la posición más a la derecha del campo de anchura *n*, según que el valor lógico en memoria sea "cierto" o "falso". Así el valor .TRUE. en memoria, empleando el código L5 se escribiría como

____T

4.6 CODIGO A

Este código se emplea para manejar y transferir información definida como tipo *carácter*. Tanto en la entrada como en la salida, las variables a transferir habrán sido definidas explícitamente, al principio del programa, mediante la sentencia de tipo CHARACTER.(Véase 1.12). La utilidad de este código se presenta cuando es necesario introducir en memoria una serie de caracteres en forma de texto, almacenándolos en variables *carácter*. Recíprocamente, cuando ha de imprimirse como texto el contenido de variables *carácter*.

La forma general de este código es

siendo A (del Inglés, Alphanumeric) el propio código y *n*, que es una constante entera sin signo y distinta de cero, la *anchura del campo*, es decir, el número de caracteres alfanuméricos a transmitir. Si *n* no figura con el código A, entonces se considera que el número total de caracteres del campo a transmitir es su anchura.

Los datos transmitidos con este código, aunque fueran numéricos, no pueden intervenir en operaciones aritméticas: no es lo mismo la constante entera 1823 que la constante *carácter* '1823'.

a) *Entrada:* Supongamos que *n*, anchura del campo, es el total de caracteres que posee el dato externamente y *m* el número de ellos que pueden ser recibidos en memoria. Este dato vendrá definido en la sentencia CHARACTER correspondiente.

Si es $m = n$, ingresa todo el dato *carácter* completamente.

Si es $n > m$, no habrá espacio para todos los caracteres en memoria. Entonces, se ignoran los $n-m$ primeros caracteres del campo de entrada y los restantes *m* se leen y se almacenan en la variable correspondiente.

Si es $n < m$, habrá espacio en exceso en memoria, de manera que ingresaran esos *n* caracteres, situándose ajustados a la izquierda de la variable y rellenándose con blancos los restantes, hasta completar los *m*.

Por ejemplo, supongamos la codificación siguiente:

```
      CHARACTER X, Y*2, Z*3
      READ (1,100) X, Y, Z
100   FORMAT (A1, A2, A3)
```

y que la línea de datos tecleada por el terminal fuera:

```
A12BCD
```

Entonces, por la primera sentencia, se sabe que X es una variable de tipo *carácter* con una longitud de 1; Y de 2 y Z de 3. Estos son los valores *m* descritos anteriormente.

Estas variables tienen reservadas en memoria, respectivamente, las anchuras 1, 2 y 3, según los códigos A de la sentencia FORMAT. Estos son los valores n de los que ya hemos hablado. Entonces, en este caso es, para cualquiera de las variables, $n = m$.

Con ello los contenidos en memoria, serían:

```
X = 'A'
Y = '12'
Z = 'BCD'
```

Si empleásemos ahora la sentencia FORMAT

```
100 FORMAT (A3, A2, A1)
```

dejando inalteradas el resto de sentencias y datos del ejemplo, tendríamos:

```
X = '2'
Y = 'BC'
Z = 'D – –'
```

La variable X tiene $n = 3$ y $m = 1$ (insuficiente) y su contenido en memoria almacenará unicamente un 2, ignorándose los $n-m = 3-1 = 2$ primeros caracteres que son la A y el 1. Para la variable Y ($n = m$) ingresa todo el campo BC. La variable Z tiene $n=1$ y $m=3$ (exceso) y su contenido en memoria ajustará a la izquierda la D y se crearan dos blancos.

Veamos ahora un ejemplo empleando el código A sin especificación de anchura.

Sea el campo a leer

```
VELOCIDAD = __ X81425
```

mediante la codificación

```
      CHARACTER X*10, B*3, C*5
      READ (1,200) X, B, C
200   FORMAT (A, A3, A6)
```

Entonces, la variable X se leería ingresando todos sus caracteres, ya que al no existir especificación de anchura del campo, ésta será el total de los caracteres que posea. Se tendría:

```
X = 'VELOCIDAD ='
B = '__X'
C = '81425_'
```

b) *Salida:* Pueden hacerse las mismas consideraciones que anteriormente.

Si $n = m$ no existe problema.

Si $n > m$, entonces el campo que sale al exterior contendrá además de los m caracteres ajustados a la derecha del campo, una serie de blancos a la izquierda hasta completarlos todos.

Si $n < m$, salen al exterior unicamente los n caracteres de más a la izquierda de la variable, perdiéndose los restantes.

Por ejemplo. Sea la codifiación

```
      CHARACTER M*2, N*4, P*1, Q*6
      WRITE (1,300) M, N, P, Q
300   FORMAT (A2, A3, A4, A5)
```

y supongamos los siguientes contenidos en memoria:

```
M = '12'        (m = 2)
N = 'ABCD'      (m = 4)
P = '8'         (m = 1)
Q = 'RESTOS'    (m = 6)
```

Entonces la salida sería:

```
12ABC___8RESTO
```

El código A puede emplearse repetido, mediante un coeficiente c, así

4.7. CODIGOS X, T, TR y TL

Estos *códigos* se denominan *de posicionamiento.* La forma general del X es

siendo *n* una constante entera sin signo y distinta de cero que indica el número total de caracteres que se van a ignorar o saltar cuando se está realizando una lectura de datos o bien el número de blancos que se van a producir cuando se está haciendo una impresión o salida de resultados.

Veamos dos ejemplos correspondientes a la entrada y salida de datos, respectivamente.

a) Supongamos una línea de terminal cuyo contenido es

```
____325 823.82______3437
    KM  ABO          JUS
```

Para leer esta línea emplearíamos una sentencia READ tal como

```
READ (1,824) KM, ABO, JUS
```

y una cualquiera de las dos sentencias FORMAT siguientes:

```
824  FORMAT (I7, F6.2, I10)
```

o bien, haciendo aplicación del código X para ignorar datos (blancos) a la entrada

```
824  FORMAT (4X, I3, F6.2, 6X, I4)
```

b) Si suponemos dos valores en memoria tales como A = 88.3 y B = 5.22, entonces empleando la codificación

```
      WRITE (1,100)
100   FORMAT (2X, F4.1, 5X, F4.2)
```

se produciría una salida por el terminal, como la siguiente:

```
__88.3_____5.22
↑ ↑        ↑ ↑
COLUMNAS 1 3      12 15
```

Para lograr la salida anterior podría emplearse también otro código: el T (del Inglés, Tabulating) con el cual se produce una tabulación. Su forma general es

siendo *n* una constante entera sin signo y no nula, que indica la posición hasta donde se quiere tabular o, lo que es lo mismo la posición del registro donde va a comenzar la transferencia de los datos. No existe en el FORTRAN básico.

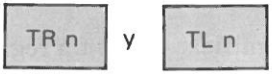
En el último ejemplo que se ha estudiado, podría producirse la misma salida de resultados empleando la sentencia FORMAT siguiente:

```
100  FORMAT (T3, F4.1, T12, F4.2)
```

Como comparación de los códigos X y T diríamos que, una especificación como 10X, por ejemplo, equivaldría a la T11 y ambas significan o bien saltar las *diez* primeras posiciones o comenzar en la posición *once*, que es la misma cosa.

Los códigos TR (del Inglés, Tabulating Right) y TL (Tabulating Left) indican a cuántos caracteres hacia la derecha o hacia la izquierda, respectivamente, a partir de la posición actual donde nos encontremos, comenzará la transferencia de los datos.

Sus formas generales son

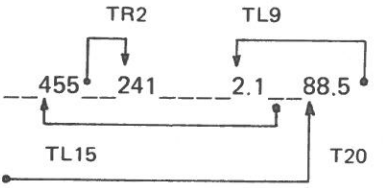


siendo *n*, como anteriormente, una constante entera sin signo y no nula.

Por ejemplo, si tenemos en memoria las variables A = 88.5, B = 2.1, I = 455 y J = 241, al emplear la codificación.

```
      WRITE (1,100) A, B, I, J
100   FORMAT (T20, F4.1, TL9, F3.1, TL15, I3, TR2, I3)
```

se produciría el siguiente registro por la pantalla del terminal:



Con flechas se indican donde comienzan y acaban las tabulaciones, así como su longitud y sentido.

Los códigos TR y TL no están incluidos en el FORTRAN básico y ninguno de los X, T, TR o TL pueden repetirse mediante el uso de un coeficiente constante, como ocurría con los anteriormente estudiados.

4.8. CODIGO DE FIN DE REGISTRO

Los códigos X y T, que acaban de estudiarse, proporcionan al programador la posibilidad de saltar o ignorar caracteres o de generar espacios en blanco, según se trate de una entrada o una salida.

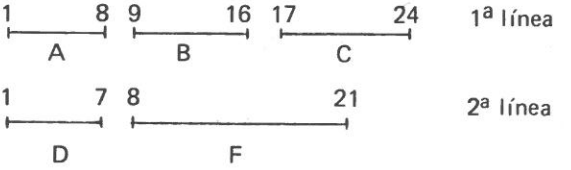
El lenguaje FORTRAN también posee la facilidad de ignorar o saltar registros completos (una ficha perforada, una línea de terminal) o bien de producir registros en blanco (una línea vacía de impresora).

Esto se logra empleando en el FORMAT el código especial,

denominado en Inglés, slash. Este código significa: fin de un registro.

Explicaremos su uso, exponiendo algunos ejemplos.

a) Supongamos que tenemos dos líneas de terminal con datos a introducir en memoria y desde las columnas que se indican:



En la primera línea existen tres variables reales A, B y C y en la segunda dos, D y F, estando esta última representada con exponente E.

Entonces para introducir en memoria esta información podríamos emplear la siguiente codificación:

```
      READ (1,100) A, B, C
100   FORMAT (3F8.2)
      READ (1,200) D, F
200   FORMAT (F7.3, E14.8)
```

Es importante observar que cada sentencia de E/S, trata un nuevo registro físico: lee una nueva línea de terminal o una nueva ficha; imprime una nueva línea de impresora o produce una nueva línea en el terminal.

El efecto de la codificación anterior puede también lograrse utilizando el código (/) así:

```
      READ (1,300), A, B, C, D, F
300   FORMAT (3F8.2, /, F7.3, E14.8)
```

lo cual significa que después de leer A, B y C el registro finaliza y las variables D y F se leerán de otro registro (de otra línea).

La sentencia FORMAT anterior también podría escribirse así:

```
300   FORMAT (3F8.2 / F7.3, E14.8)
```

suprimiendo las comas que precede y sigue al slash.

b) Se trata de visualizar por la pantalla del terminal cuatro variables, AA, II, BB, JJ contenidas en memoria. Al emplear

```
      WRITE (1,100) AA, II, BB, JJ
100   FORMAT (F5.2, I4 / F6.3, I2)
```

obtendríamos dos registros o dos líneas sucesivas así:

```
AA      II      ← 1ª línea
(F 5.2) (I4)

BB      JJ      ← 2ª línea
(F6.3)  (I2)
```

c) El mismo ejemplo anterior pero empleando como FORMAT

```
100  FORMAT (F5.2, I4 // F6.3, I2)
```

obtendríamos un salto múltiple.

```
AA        II      ← 1ª línea
(F5.2)    (I4)

                  ← 2ª línea en blanco

BB        JJ      ← 3ª línea
(F6.3)    (I2)
```

Los dos slashes dan lugar a que se produzcan dos altos sucesivos de línea o, lo que es igual, que se deje una línea en blanco entre los dos registros.

El empleo del slash mostrado en el último ejemplo para saltos múltiples puede generalizarse empleando varios slashes. Pero según que estén situados al principio, en medio o al final de la sentencia FORMAT, el efecto producido conviene aclararlo. Las reglas a tener en cuenta en cada caso vienen representadas en el siguiente cuadro:

NUMERO DE SLASHES /	SITUACION DENTRO DEL FORMAT	REGISTROS IGNORADOS O LINEAS EN BLANCO
n	Al principio de la sentencia	n
n	En el interior de la sentencia	n-1
n	Al final de la sentencia	n

Cuadro 4.1

Veamos otros ejemplos:

d) Si deseamos leer tres variables reales de una ficha perforada perteneciente a un paquete de ellas, ignorando o saltando las cinco

primeras fichas, deberíamos emplear una sentencia de formato como la siguiente:

```
FORMAT (///// 3F6.2)
```

e) La sentencia

```
FORMAT (F5.2 /// I4)
```

empleada para salida de resultados sobre una pantalla, produciría dos líneas de información, con dos líneas en blanco entre ellas, es decir, trata cuatro registros físicos, dos de los cuales son dos líneas en blanco.

f) La sentencia

```
FORMAT (I3, E14.8 ////)
```

escribiría una línea, por ejemplo, con dos variables y luego saltaría las cuatro líneas siguientes.

4.9. CODIGOS PARA CONSTANTES CARACTER

Existen dos códigos especiales para la salida de constantes tipo *carácter:* son los códigos H (del inglés, Hollerith) y apóstrofe ('), este último ya utilizado anteriormente para definir y usar las variables *carácter* (véanse, por ejemplo, 1.8 y 3.5).

La forma general del código H es

siendo *n* una constante entera sin signo y no nula. Este código da lugar a que salgan *n* caracteres alfanuméricos (incluidos los blancos) al exterior.

Por ejemplo, para producir una línea en el terminal con el mensaje PARADA DEFINITIVA, se escribiría:

```
      WRITE (1,100)
100   FORMAT (17H PARADA DEFINITIVA)
```

Es mucho más cómodo emplear para el mismo fin el apóstrofe. Así, con el uso de éste, el ejemplo anterior se convertiría en

```
      WRITE (1,100)
100   FORMAT  (17HPARADA DEFINITIVA)
```

Para producir la salida de los contenidos de dos variables A y B, precedidos de una constante *carácter*, podría emplearse, por ejemplo, el par de sentencias siguientes:

```
      WRITE (1,200) A, B
200   FORMAT ('ABSCISA =' F4.1,'_ORDENADA =' F4.2)
```

las cuales producirían por el terminal un registro como el

```
ABSCISA = xx.x _ORDENADA = x.xx
          ~~~~            ~~~~
            A               B
```

4.10. CONTROL DE CARRO

El lenguaje FORTRAN dispone de códigos especiales para controlar el carro de la impresora, si se ha elegido este dispositivo como medio de salida de los datos. Los resultados a imprimir son colocados todos juntos y enviados luego a la impresora línea a línea.

Realmente no se trata de códigos de control sino más bien caracteres empleados como códigos. Estos caracteres se indican en el cuadro adjunto y deben escribirse como primer carácter dentro de la sentencia FORMAT asociada con otra WRITE que emplee la impresora para la salida de resultados.

CARACTER DE CONTROL	EFECTO SOBRE LA IMPRESORA
blanco	El papel avanza una línea antes de comenzar la impresión
0	El papel avanza dos líneas antes de comenzar la impresión
1	Avanzar el papel hasta nueva página, en las impresoras de alta velocidad
+	Imprimir de nuevo sobre la misma línea (superposición)

Cuadro 4.2

Ejemplos:

a) Las sentencias

```
      WRITE (2,100)
100   FORMAT ('1', 5X, 'PROGRAMA DE CONTROL')
```

darían lugar a que se imprimiera (unidad número 2) el mensaje PROGRAMA DE CONTROL precedido por cinco blancos, pero ello una vez que el papel de la impresora se hubiera situado en una nueva página.

b) Si el FORMAT del ejemplo anterior, correspondiente a la salida por impresora, fuera

```
100   FORMAT (5X, 'PROGRAMA DE CONTROL')
```

entonces se tomaría como carácter de control el primer blanco de los cinco definidos por 5X. Con ello se producirían dos efectos: el papel avanzaría una línea antes de comenzar la impresión (se imprimiría en la línea siguiente) y el cabecero o mensaje vendría precedido únicamente por cuatro blancos,así

```
____PROGRAMA_DE_CONTROL
```

La regla práctica a seguir es: escribir siempre el carácter de control como constante *carácter* y no implícitamente aprovechando otro código.

Obsérvese que '1' en el ejemplo a) sería equivalente a 1H1 y '0' equivaldría a 1H0 y a 1X.

4.11. CODIGOS SP, SS y S

Estos códigos únicamente tienen efecto en la salida de resultados del tipo I, F, E, G y D y no están incluidos en el FORTRAN básico.

La impresión o la salida de datos numéricos positivos, normalmente, se efectúa sin la producción de un signo + delante del primer dígito.

Pero para controlar la salida de ese signo opcional, ya que internamente en memoria acompaña al dato (véase 4.1 a) y b)), se emplean los códigos SP, SS y S, cuyo significado es:

SP: producir un signo + en cualquiera de las subsiguientes posiciones que normalmente contengan tal signo como opcional, es decir, en los valores numéricos positivos que siguen.

SS: no producir el signo + en ninguna de las subsiguientes posiciones que normalmente contienen tal signo como opcional. Se producirán espacios en su lugar.

S: Se restaura la opción de producir signo + .

Por ejemplo, la sentencia

```
PRINT '(I3, SP, F4.1, A)', J, AL, NOMB
```

producirá la impresión de un signo + delante del contenido de la variable AL, cuando éste sea positivo.

Resumiendo: cuando en una sentencia de formato correspondiente a una operación de salida de resultados, aparecen los códigos SP, SS, S, los campos numéricos que les siguen en el FORMAT quedan afectados respecto a la utilización delante de ellos del signo +, de acuerdo con las reglas enunciadas más arriba.

Otro ejemplo:

```
PRINT '(I2, SP, F4.2, E14.8, S, I5, A)', J, ALA, BE, K, NOMB
```

produciría un signo + delante del contenido de las variables ALA y BE (si son positivas) pero no delante del contenido de la K aunque fuera positiva.

4.12. CODIGOS BN y BZ

Estos códigos afectan únicamente a la entrada de datos numéricos del tipo I, F, E, G y D.

Cuando se leen datos numéricos, es normal que los espacios en blanco a la izquierda de cada uno de ellos, se ignoren. Si un campo de entrada es completamente blanco se considera como cero. Ahora bien; los blancos entre dos dígitos o al extremo derecho del campo, normalmente se ignoran.

Por ejemplo, la codificación

```
      READ (1,100) M, N, L, J
100   FORMAT (I3, I5, I4, I3)
```

sobre la línea de datos en un terminal

```
___13_5724_625_
  M   N   L   J
```

realizará la siguiente asociación de variables y valores:

```
M = 0
N = 1357
L = 246
J = 25
```

Ahora bien; esta forma de interpretar los datos puede alterarse, empleando los códigos BN y BZ:

BN significa ignorar los blancos (blanks null)
BZ significa tratar los blancos como ceros (blanks zero)

Así, si el ejemplo anterior tuviera la sentencia de formato

```
100  FORMAT (I3, BZ, I5, I4, BN, I3)
```

la asociación de variables y valores, sería ahora

```
M = 0
N = 13057
L = 2406
J = 25
```

El código BZ afecta a la variable N y a todas las subsiguientes (en este caso la L) hasta que aparezca un código BN.

Recuérdese (véase 3.13, 8) que al comienzo de la operación de entrada a un fichero, se establece un control, determinado por las distintas opciones de la sentencia OPEN y, concretamente, la que afecta a este código es la

```
BLANK = bl
```

Cuando esta opción no figura en la sentencia OPEN, es como si se tratase de la BLANK = 'NULL', que equivale al código BN.

4.13. CODIGO P

Este código sirve para indicar un *factor de escala n* y es aplicable únicamente a datos no enteros, es decir, afectados por los códigos F, E, G y D.

Su forma general es

siendo *n* una constante entera que, opcionalmente, puede llevar signo.

El sistema considera que el factor de escala *n* es siempre cero mientras no se establezca otra cosa empleando el código P.

a) *Entrada:* Cuando los datos a ser transferidos a memoria vienen ya acompañados de exponente, el factor de escala *n* no tiene ningún efecto. En caso contrario, la relación entre el valor externo y el interno es

$$\text{valor interno} = \frac{\text{valor externo}}{10^n}$$

Supongamos por ejemplo que tenemos la codificación

```
      READ (1,100) A, B
100   FORMAT (1PF6.1, -1PF5.0)
```

y que el registro de entrada fuera:

```
12.34567.89
 ~  ~  ~  ~
 A     B
```

Entonces los valores ingresados en memoria serían:

$$A = \frac{12.345}{10^1} = 1.2345$$

$$B = \frac{67.89}{10^{-1}} = 678.9$$

b) *Salida:* Cuando el factor de escala se utiliza en conexión con un formato F, entonces la relación entre valores externos e internos es

valor externo = valor interno x 10^n

Cuando P se emplea en unión con los códigos E o D, el valor no cambia, pero el punto decimal se desplaza *n* lugares a la derecha, si *n* es positivo, en el resultado obtenido, mientras que el exponente se reduce también en tantas unidades como indica *n*.

Utilizando P con el código G, el valor no se altera; pero si el resultado obtenido se produce en la forma mantisa-exponente, tanto exponente como punto decimal de la mantisa siguen las reglas enunciadas en el párrafo anterior para los códigos E o D.

Por ejemplo, el valor interno 0.135792, con la expecificación 3PF7.4, produciría como resultado:

$135.792 = 0.135792 \times 10^3$

Si el valor interno 0.13579×10^3, se imprimiera con E9.3, el resultado sería

```
0.136E+03
```

Pero si se imprimiera con 1PE9.3, se obtendría

```
1.357E+02
```

El factor de escala puede afectar a más de un código numérico. Por ejemplo,

```
2P, 3F7.4
```

es un factor de escala 2 que afecta a *tres* códigos F. Obsérvense, en este caso, la coma (,) que sigue a la P.

4.14. CODIGO (:)

Este código, utilizado en una sentencia FORMAT en conexión con una lista de datos, produce el efecto de finalizar la transmisión de los mismos si no existen más variables en la lista de E/S.

Por ejemplo, supongamos la siguiente codificación:

```
      I1 = 101
      I2 = 2574
10    WRITE (1,100) I1, I2
      I3 = 285
      I4 = 10
      A1 = 3.28
20    WRITE (1,100) I3, I4, A1
100   FORMAT ('I =', I3, ' J =', I4, :,' A =', F4.2)
```

Entonces, por la primera sentencia WRITE número 10 se produciría, en conexión con la sentencia de formato número 100, la siguiente salida por la pantalla:

```
I = 101_ J = 2574
```

Al llegar el programa a la sentencia 20 WRITE, la salida sería:

```
I = 285_J = __10_A = 3.28
```

Este código no figura en el FORTRAN básico.

4.15. USOS DIVERSOS DEL FORMAT

A continuación vamos a exponer una serie de ejemplos y casos prácticos donde se analizarán diversos usos de la sentencia FORMAT. Se trata, podríamos decir, de una serie de reglas generales importantes y muy prácticas que deben entenderse con claridad.

Es frecuente que un grupo de códigos diferentes se repita un cierto número de veces. En este caso basta encerrar entre paréntesis tal grupo, haciéndole preceder de un coeficiente o constante entera positiva igual al número de veces que va a repetirse.

Ejemplo:

El grupo de especificaciones I2, E10.4, F8.2 se repite cuatro veces. Entonces bastaría con escribir dentro de la sentencia FORMAT

```
4 (I2, E10.4, F8.2)
```

y ello valdría para el tratamiento de 12 variables leídas o escritas.

Por lo expuesto hasta el momento, puede parecer que cada sentencia FORMAT debería contender tantos códigos de formato como datos existan en la correspondiente lista de la sentencia de E/S que va ligada con dicho FORMAT. En los ejemplos expuestos hasta aquí así sucedía; pero es posible que ambas cantidades (variables tratadas y códigos de formato) no coincidan.

De no coincidir ambos números pueden darse dos casos:

1. El número de datos de la lista de E/S, es menor que el de códigos de formato, en cuyo caso los códigos sobrantes se ignoran.

Ejemplo:

Si los datos A y B se transmiten mediante la sentencia de formato,

```
100  FORMAT (F8.2, E10.4, F8.3, I4)
```

no se tendrían en cuenta los dos últimos códigos F8.3 e I4.

2. El número de datos de la lista de E/S es mayor que el de códigos de formato, en cuyo caso el control de la sentencia FORMAT se transfiere al paréntesis de apertura precedente, empleándose de nuevo los mismos códigos de formato para los siguientes datos de la lista, pero estos leídos o escritos de/en registros físicos diferentes. Lo dicho se refiere al caso de que en el FORMAT exista un único nivel de paréntesis.

Ejemplos:

a) Las siguientes sentencias

```
      WRITE (1,100), A, B, C, D, E, F, G
100   FORMAT (F8.3, E10.4)
```

darían lugar a que se visualizaran las variables A, B, ..., G, en la pantalla del terminal según el orden y disposición siguientes:

1ª línea	Datos	A	B
	Formatos	F8.3	E10.4
2ª línea	Datos	C	D
	Formatos	F8.3	E10.4

3ª línea	Datos	E	F
	Formatos	F8.3	E10.4
4ª línea	Datos	G	
	Formatos	F8.3	

Con ello podría concluirse diciendo que la lista de variables de una sentencia de E/S se satisface siempre, es decir, se lee, escribe o perfora completamente. Además, la reexploración de los códigos que componen una sentencia de formato lleva consigo el cambio de un registro físico al siguiente. En el ejemplo que precede, de una línea a otra.

b) Utilizando para esos mismos datos anteriores la sentencia de formato,

```
100 FORMAT (F8.3/E10.4,F8.2)
```

los datos se visualizarían así:

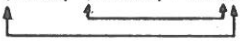
1ª línea	Datos	A	
	Formatos	F8.3	
2ª línea	Datos	B	C
	Formatos	E10.4	F8.2
3ª línea	Datos	D	
	Formatos	F8.3	
4ª línea	Datos	E	F
	Formatos	E10.4	F8.2
5ª línea	Datos	G	
	Formatos	F8.3	

Si en el FORMAT existe más de un nivel de paréntesis, el primer registro corresponde a todos los códigos existentes dentro del nivel más externo y los siguientes se toman desde el nivel más cercano al paréntesis de cierre del nivel primero, hasta el final.

Ejemplos:

a) Se trata de visualizar por la pantalla del terminal una serie de datos, empleando para ello la sentencia FORMAT con dos niveles de paréntesis

```
100  FORMAT (F8.3,  (E10.2, F6.3))
```



Las líneas contendrían los siguientes formatos, cada uno de los cuales correspondería a una determinada variable:

```
F8.3, E10.2, F6.
E10.2, F6.3
E10.2, F6.3
. . . . . .
. . . . . .
```

b) Si la sentencia empleada fuera

```
100  FORMAT  ((F8.3, E10.2), F6.3)
```

las líneas se visualizarían así:

```
F8.3, E10.2, F6.3
F8.3, E10.2, F6.3
  . . . . . .
  . . . . . .
```

c) Al emplear

```
100  FORMAT (I2/(I3, F6.1), F9.7)
```

se producirían las siguientes líneas, de las que se indican los formatos de las variables que contendrían:

```
I2
I3, F6.1, F9.7
```

```
I3, F6.1, F9.7
. . . . . .
. . . . . .
```

d) Empleando

```
100  FORMAT ((I2, I3), (I5, F8.2), F9.3)
```

las líneas con sus formatos, serían:

```
I2, I3, I5, F8.2, F9.3
I5, F8.2, F9.3
I5, F8.2, F9.3
. . . . . .
. . . . . .
```

e) Sea ahora el formato

```
100  FORMAT (2 (I4, F5.0))
```

entonces los formatos y registros de las variables serían:

```
I4, F5.0, I4, F5.0
I4, F5.0, I4, F5.0
. . . . . .
```

f) Si el formato fuera

```
100  FORMAT (2 (I4, F5.0), (I3, F4.0))
```

la salida sería:

```
I4, F5.0, I4, F5.0, I3, F4.0
I3, F4.0
I3, F4.0
. . . . . .
```

EJERCICIOS RESUELTOS

4.1. Se tienen los siguientes campos: K1 = 384; K2=-337; K3 = 4583 y K4 = 3. Están tecleados uno a continuación de otro sobre un termimal.

Escribir los códigos adecuados para poderlos leer en memoria.

Respuesta

K1 → I3
K2 → I4
K3 → I4
K4 → I1

4.2. Escribir una sentencia FORMAT adecuada para poder interpretar los datos del ejercicio anterior.

Respuesta

FORMAT (I3, I4, I4, I1) o bien
FORMAT (I3, 2I4, I1)

4.3. Se tienen en memoria las siguientes cantidades: 250, −1820, 43 y −7450, todas ellas enteras. Escribir la sentencia FORMAT correspondiente, de manera que salgan al exterior esos datos con dos blancos entre cada uno de ellos.

Respuesta

FORMAT (I3, I7, I4, I7) produciría la salida

```
250__−1820__43__−7450
 I3     I7    I4    I7
```

4.4. Se tienen en memoria las cantidades siguientes: −378, 4709 y −47 todas ellas enteras. Decir la salida que se produce al emplear cada una de las siguientes sentencias FORMAT:

a) FORMAT (2I4, I3)
b) FORMAT (2I6, I5)
c) FORMAT (I5, I6, I6)
d) FORMAT (I3, I5, I2)

Respuesta

a) −3784709−47
b) __−378__4709__−47
c) _−378__4709___−47
d) ***_4709**

4.5. Se tiene el siguiente programa parcial:

```
      I = -820
      JJ = 1450
      WRITE (1,200) I, JJ
200   FORMAT (I6, I5)
```

Decir cuál será la salida producida.

Respuesta

__-820_1450

4.6. Supongamos que una línea de terminal contiene la siguiente información:

_203_4-825__72483

Para leer esta información se ha empleado en el programa la opción BLANK = 'ZERO' y las sentencias de formato que se indican. ¿Cuál será en cada caso la información ingresada.

a) FORMAT (I2, I4, I5, I6)
b) FORMAT (I4, I2, I5)
c) FORMAT (2I6, I5)

Respuesta

a) 2, 304, -8250, 72483
b) 203, 4, -8250
c) 20304, -82500, 72483

4.7. Dada la siguiente relación de valores enteros internos y códigos respectivos de salida, decir cuáles serán los valores obtenidos en el soporte externo.

	VALOR INTERNO	CODIGO
a)	-1253	I10.6
b)	824	I6.3
c)	8725	I9.6
d)	-10720	I11.8

Respuesta

a) ___-001253
b) ___824 (Este código equivale a I6)

c) ___008725
d) __–00010720

4.8. Se tienen en memoria las siguientes variables y contenidos:

ALTU = 0.3875×10^{2}
BASE = 0.447×10^{-1}
TOT = -0.845×10^{1}

Se trata de visualizarlas en una pantalla empleando las siguientes sentencias FORMAT:

a) FORMAT (F5.2, F6.4, F5.2)
b) FORMAT (F5.2, 2 F7.4)
c) FORMAT (F7.2, F8.4, F6.2)

Escribir la salida producida en cada caso

Respuesta

$0.3875 \times 10^{2} \equiv 38.75$,, $0.447 \times 10^{-1} \equiv 0.0447$,, $-0.845 \times 10^{1} \equiv -8.45$

a) 38.750.0447–8.45
b) 38.75_0.0447–8.4500
c) __38.75__0.0447__–8.45

4.9. Se tienen las variables y valores correspondientes siguientes:

VARIABLE	VALOR
ALU	–347.82
IKO	1723
SOSP	83.723
ALFA	44.23
IX	32
IY	43

Estas variables se introducen por un terminal en memoria sin escribir el punto decimal. Deducir las sentencias FORMAT correspondientes a las READ siguientes:

a) READ (1,101) ALU, IX, IY
b) READ (1,308) IKO, IX, IY

c) READ (1,1000) ALU, IKO, SOSP
d) READ (1,1724) ALU, IKO, SOSP, ALFA, IX, IY

Respuesta

a) 101 FORMAT (F6.2, I2, I2) o bien
   101 FORMAT (F6.2, 2I2)
b) 308 FORMAT (I4, 2I2)
c)1000 FORMAT (F6.2, I4, F5.3)
d)1724 FORMAT (F6.2, I4, F5.3, F4.2, 2I2)

4.10. En memoria están contenidas las siguientes variables cuyos valores se indican:

VARIABLE	VALOR NORMALIZADO Y EQUIVALENTE		
KMS	–		325
X	0.833×10^{1}	≡	8.33
Y	0.237×10^{-1}	≡	0.0237
JOTA	–		–47
ABO	0.7834×10^{3}	≡	783.4

Escribir las sentencias FORMAT correspondientes a las WRITE que se indican, teniendo en cuenta que en la salida de resultados por el terminal se dejarán dos blancos entre cada dos campos.

a) WRITE (1,10) KMS, JOTA, ABO
b) WRITE (1,20) X, Y, KMS
c) WRITE (1,30) KMS, X, Y, JOTA, ABO

Respuesta

a) 10 FORMAT (I3, I5, F7.1)
b) 20 FORMAT (F4.2, F8.4, I5)
c) 30 FORMAT (I3, F6.2, F8.4, I5, F7.1)

4.11. En memoria están contenidas las variables reales A=-3.22 y B = 87.40. Qué salida se produce con las sentencias siguientes:

```
      WRITE (1,127) A, B
127   FORMAT (F4.2, F5.2)
```

Respuesta

****87.40

4.12. Para leer en memoria una serie de valores externos, se emplea el código E10.3. Encontrar en cada caso el valor almacenado internamente.

a) __3.235+03
b) ___4321+02
c) 3.8324E+01
d) __−374.1E+3
e) 321876E+02

Respuesta

a) 0.3235×10^4
b) 0.4321×10^3
c) 0.38324×10^2
d) Especificación insuficiente. El campo ocupa once posiciones.
e) 0.321876×10^5

4.13. Decir el resultado impreso, empleando E12.3, para los valores en memoria siguientes:

a) 0.3583678×10^3
b) 0.3700000×10^5
c) -0.8247000×10^0
d) 0.3457824×10^{-1}

Respuesta

a) ___0.358E+03
b) ___0.370E+05
c) __−0.825E+00
d) ___0.346E−01

4.14. Idem que los datos anteriores empleando E15.3E4

Respuesta

a) ____0.358E + 0003
b) ____0.370E + 0005
c) ___− 0.825+0000
d) ____0.346E

4.15. Idem que 4.13 empleando D9.3

Respuesta

a) 0.358D + 03
b) 0.370D + 05
c) *********
d) 0.346D−01

4.16 Se emplea el código G11.6 para imprimir los siguientes datos internos:

a) 0.246891×10^2
b) -0.135791×10^{-1}
c) 0.12345×10^3
d) 0.424381×10^3

Encontrar las salidas impresas.

Respuesta

a) 24.6891____
b) Espeficiación insuficiente
c) 123.450____
d) 424.381____

4.17. Se tiene la siguiente lista de variables, definidas como *carácter*, con sus respectivos tamaños y valores en memoria:

VARIABLE	TAMAÑO m	VALOR	CODIGO
X	5	'ABCDE'	A7
Y	2	'87'	A2
Z	9	'S = 1325780'	A5
V	4	'1331'	A6
W	3	'M__'	A1

Decir cuales serían los valores impresos empleando los códigos expresados en la tabla.

Respuesta

X → __ABCDE
Y → 87
Z → S = 132
V → __1331
W → M

4.18. Por el terminal se introduce el siguiente conjunto de caracteres:

1357924680ABCDEFGHIJK

empleando la codificación

```
      CHARACTER*4   X, Y, Z * 3, V * 2, W * 4
      READ (1,100) X, Y, Z, V, W
  100 FORMAT (A4, A3, A5, A2, A)
```

Encontrar los contenidos de cada variable en memoria

Respuesta

X = '1357'
Y = '924_'
Z = 'OAB'
V = 'CD'
W = 'EFG'

4.19. Mediante un terminal se introduce el siguiente conjunto de caracteres:

1234567___90__·TRABAJO___FIN

para lo cual se ha empleado en el programa una codificación FORTRAN como la siguiente:

```
      CHARACTER*2  X, Y, Z, V, W * 5
      LOGICAL A, B
      READ (1,200) X, Y, Z, V, W, A, B
200   FORMAT (2A2, A1, A2, A5, L10, L4)
```

Encontrar los contenidos de cada variable dentro de la memoria.

Respuesta

X = '12'
Y = '34'
Z = '5_'
V = '67'
W = '___90'
A = ·TRUE·
B = ·FALSE·

4.20. Se tienen almacenados en memoria los valores *cierto* y *falso*, respectivamente, en las variables definidas como *lógicas.*

A = ·TRUE·
B = ·FALSE·

Encontrar la salida impresa al emplear los códigos L2, L5 y L1 para la A y los L1, L3, L4 para la B.

Respuesta

a) Para la A

_T
____T
T

b) Para la B

F
__F
___F

4.21. Se tiene una línea de terminal cuyo contenido es:

5834 __ 153.57 ____ 123.2152 3.235
K A B J C

De este registro únicamente se desea ingresar en memoria los campos correspondientes a las variables K, B (sin leer los blancos de la izquierda) y C. Escribir unas sentencias READ y FORMAT que produzcan este objetivo.

Respuesta

Una solución sería:

```
      READ (1,100) K, B, C
100   FORMAT (I4, 12X, F5.0, 3X, F5.0)
```

4.22. Dados los valores almacenados en memoria

A = 152.830
I = 533
B = 1.2357

Encontrar cuál sería la impresión producida (a partir de la columna uno) empleando cada una de las sentencias FORMAT siguientes:

a) 100 FORMAT (2X, F7.3, 2X, I3, 3XF6.4)
b) 200 FORMAT (F7.3, 4X, I3, 1X F6.4)

Respuesta

a) __152.830–_533___1.2357
b) 152.830____533_1.2357

4.23. Escribir las dos sentencias FORMAT anteriores, empleando el código T en lugar del X, para que ambos produzcan el mismo efecto.

Respuesta

a) 100 FORMAT (T3, F7.3, T12, I3, T18, F6.4)
b) 200 FORMAT (F7.3, T12, I3, T16, F6.4)

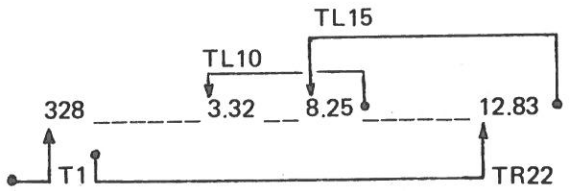
4.24. Se tienen almacenados en memoria los siguientes valores:

KM = 328
VEL = 12.83
X = 8.25
Y = 3.32

Describir la salida impresa que se producirá al emplear la codificación siguiente:

```
      WRITE (1,200) KM, VEL, X, Y
200   FORMAT (T1, I3, TR22, F5.2, TL15, F4.2, TL10, F4.2)
```

Respuesta



4.25. Se tienen en memoria las coordenadas X e Y de tres puntos distintos, cuyos valores son, respectivamente:

X(1) = 3.22 ,, Y(1) = –4.35
X(2) = –2.83 ,, Y(2) = 9.70
X(3) = 5.21 ,, Y(3) = 2.59

Haciendo uso de la sentencia DO, imprimir por el terminal, cada pareja de valores en una línea distinta y dejando entre cada dos, una en blanco. Escribir el FORMAT utilizando los códigos H o bien títulos entre comillas, para que delante de cada coordenada salga el mensaje X=, y el Y=.

Respuesta

```
      DO 12 I = 1,3
12    WRITE (1,100) I, X(I), Y(I)
```

y luego una cualquiera de las dos sentencias:

a) 100 FORMAT ('PUNTO', I2, 'X=', F5.2, 'Y=', F5.2/)
b) 100 FORMAT (5HPUNTO, I2, 2HX=, F5.2, 2HY=, F5.2/)

4.26. Supónganse las variables y contenidos siguientes:

A = 130.83
B = −43.859
C = 2.13
II = 80
JJ = 913

Dedúzcanse los resultados obtenidos al emplear las sentencias siguientes:

a) PRINT '(I4, F8.2, SP, F8.2, I4)', II, B, A, JJ
b) PRINT '(SP, I5, F9.3, SS, F9.3, I5)', II, A, C, JJ
c) PRINT' (SP, I5, SS, F9.3, S, F9.2, I5)', II, A, C, JJ

Respuesta

a) __80__−43.85_+130.83 + 913
b) __+80__+130.830____2.130__913
c) __+80__130.830____+2.13_+913

4.27. El siguiente registro de entrada, formado por los campos que se indican, se introduce en memoria a través de un terminal:

20_2723.7_941__2_37__

II A B JJ

mediante las correspondientes instrucciones FORMAT:

a) 100 FORMAT (BZ, I5, F6.3, F6.1, BN, I5)
b) 100 FORMAT (I5, F6.3, BZ, F6.1, I5)
c) 300 FORMAT (I5, F6.3, F6.1, I5)

Deducir los valores ingresados en memoria en cada caso.

Respuesta

a) II = 20027
A = 23.709
B = 4100.2
JJ = 37

b) II = 2027
A = 23.79
B = 4100.2
JJ = 3700

c) II = 2027
A = 23.79
B = 41.2
JJ = 37

4.28. A través de un terminal se introduce el siguiente registro formado por los campos A, B, X, II:

342.86 _ 15.235 _ 83.2E+04 _ 1984

A B X II

Para ello, las correspondientes instrucciones FORMAT serían:

a) 100 FORMAT (2PF6.0, –2PF7.3, 3PE9.1, I5)
b) 200 FORMAT (F6.0, 4PF7.0, E9.0, I5)

Deducir en ambos casos cuál será el contenido en memoria de cada variable.

Respuesta

a) A = 3.4286
B = 1523.5
X = 832000.0
II = 1984

b) A = 342.86
B = 0.0015235
X = 832000.0
II = 1984

4.29. Deducir el orden en que se produciría la salida por un terminal de los campos indicados en la sentencia WRITE siguiente (indicar campos y formatos):

```
      WRITE (1,200) A, B, X, KM, KS, C, D
200   FORMAT (3F8.2, I4, I3)
```

Respuesta

Línea 1ª A B C KM KS
F8.2 F8.2 F8.2 I4 I3

Línea 2ª C D
F8.2 F8.2

5

Elementos lógicos. Estructuras de Control

Fe que no duda es fe muerta

M. DE UNAMUNO *(1864-1937)*

5.1. OPERADORES Y EXPRESIONES DE RELACION

En el primer capítulo de este libro introdujimos la noción de dato lógico (1.7 y 1.11). Por tal entendiamos las constantes lógicas

. TRUE .
. FALSE .

y las variables lógicas que están definidas explícitamente mediante la sentencia de tipo LOGICAL.

Vamos ahora a ampliar el campo de los datos lógicos estudiando otros conceptos FORTRAN relacionados con los mismos.

Se denomina *operador de relación* un operador tal que al ser aplicado sobre dos elementos del mismo tipo, datos numéricos o *carácter*, produce un resultado lógico, es decir, *cierto* o *falso*. Estos operadores, en FORTRAN, son los siguientes:

OPERADOR DE RELACION	SIGNIFICADO
· GT ·	Mayor que (>)
· GE ·	Mayor o igual que (≥)
· LT ·	Menor que (<)
· LE ·	Menor o igual que (≤)
· EQ ·	Igual que (=)
· NE ·	Distinto de (≠)

Las seis palabras clave anteriores deben ir precedidas y seguidas por un punto (.).

Veamos ahora un ejemplo de su uso. El discriminante de la ecuación polinómica de segundo grado tiene la forma

$$b^2 - 4\,ac$$

y es muy común comparar este valor con cero para saber que tipo de raíces posee dicha ecuación. Utilizando el operador de relación .GE. esta comparación podría escribirse, en FORTRAN, como

```
( B**2 - 4. * A * C). GE . 0
```

lo que constituye una *expresión de relación*. En este caso, esa expresión se ha formado aplicando .GE. a las expresiones aritméticas

```
B ** 2 - 4. * A * C
```

y a la

```
0  (cero)
```

obteniéndose como resultado un valor lógico, cierto o falso.

Sería como preguntar: ¿es el valor del discriminante mayor o igual a cero? Una contestación afirmativa equivale a *cierto;* una negativa, a *falso*.

Como ya se ha dicho anteriormente, los operadores de relación se aplican no sólo a expresiones aritméticas sino también a expresiones *carácter* que serán estudiadas más adelante. No pueden aplicarse, sin embargo a una expresión aritmética y a otra *carácter*.

Los operadores .EQ. y .NE. pueden aplicarse, además, a una expresión aritmética y a otra *expresión compleja*. La aritmética de los *números complejos* se estudiará en otro apartado.

Veamos a continuación algunos ejemplos. Supóngase una serie de variables cuyo tipo es el que se indica:

NOMBRE DE VARIABLE	ESPECIFICACION O TIPO
E, VARI	Reales
I, J, KILO	Enteras
R	Lógica
C	Compleja
NOMBRE	Carácter

Las expresiones de relación siguientes son válidas o inválidas, según se indica:

E . LT . I	Válida. Pueden mezclarse tipos enteros y reales.
VARI ** 2 . GE . 27.7E+5	Válida.
C . EQ . 3 * J	Válida.
C . GT . KILO	Inválida. El operador .GT. no puede aplicarse a una cantidad compleja.
. LE . 5	Inválida. Falta una expresión delante del operador de relación.
NOMBRE . EQ . 'LIBRO'	Válida.
R . NE . (E + KILO)	Inválida. Una variable lógica, R, no puede ser tratada por un operador de relación.
VARI . NE . 'XYZ'	Inválida. La primera expresión es real y la segunda es una constante *carácter*.

5.2. OPERADORES LOGICOS

Con el fin de poder formar expresiones lógicas más complejas que las estudiadas hasta el momento, el lenguaje FORTRAN dispone de unos operadores especiales, denominados *operadores lógicos*. Con ellos se amplian las posibilidades del álgebra lógica o álgebra de Boole. Estos operadores son cinco:

. NOT .
. AND .
. OR .
. EQV .
. NEQV .

y los tres primeros equivalen, respectivamente, a las operaciones booleanas básicas de *negación* (NO), *multiplicación* o *intersección* (Y) y *suma* o *unión* (O). Los otros dos significan, respectivamente, *equivalente* y *no equivalente*.

Obsérvese que los operadores lógicos también están acompañados de puntos (.).

Cada uno de estos operadores, excepto el ·NOT· se aplica a una expresión lógica FORTRAN, es decir, si designamos por *op* estos operadores y por *expre log* una expresión lógica FORTRAN, la forma general sería:

```
(expre log1) . op . (expre log2)
```

Para .NOT.. la forma general es

```
.NOT. (expre log)
```

Veamos el uso y significado de estos operadores lógicos mediante algunos ejemplos:

a) La expresión FORTRAN

```
.NOT. A
```

significa lo siguiente: A debe ser una expresión lógica que podrá tomar los valores cierto o falso. Entonces, si A es cierto la expresión .NOT.A es falsa; si A es falso, la expresión será cierta.

Es decir, el operador .NOT. produce una negación, como ya se había definido anteriormente.

b) La expresión

```
A. AND . B
```

significa: si A y B son ambos ciertos, la expresión será cierta; si ambos son falsos, será falsa; basta que una de ellas sea falsa para que la expresión total también lo sea

c) La expresión

```
A . OR . B
```

significa: si A y/o B son ciertos, la expresión total será cierta; si ambos son falsos, la expresión anterior tomará el valor falso.

En el cuadro siguiente, se resume el resultado obtenido al emplear cada uno de los operadores lógicos. En este cuadro, A y B son expresiones lógicas FORTRAN; C significa cierto y F falso.

SI ES	A	F	F	C	C
	B	F	C	F	C
ENTONCES SERA	· NOT · A	C	C	F	F
	A · AND · B	F	F	F	C
	A · OR · B	F	C	C	C
	A · EQV · B	C	F	F	C
	A · NEQV · B	F	C	C	F

Cuadro 5.1

Veamos algunos ejemplos más. Supongamos que A, B y C son tres variables lógicas FORTRAN, con los valores respectivos siguientes:

```
A = . FALSE .
B = . TRUE .
C = . TRUE .
```

a) La expresión

```
D = B . AND . A . OR .. FALSE .
```

es falsa, es decir, será D = . FALSE .

b) La expresión

```
E = A . OR . (B . AND . C)
```

es cierta, es decir, será E = . TRUE . . Obsérvese que en primer lugar se evalua la expresión dentro del paréntesis, obteniéndose como resultado . TRUE . y luego se aplica el operador . OR . a la expresión obtenida y a la A.

c) La expresión

```
F = (A . OR . C) . AND . (. NOT . B)
```

es falsa, porque A. OR . C es cierto y . NOT . B es falso. Como consecuencia, será F = . FALSE . .

Los operadores lógicos pueden emplearse uno a continuación de otro, es decir, en secuencia, sin ninguna expresión lógica entre ellos; pero esta secuencia está sometida a unas reglas, de modo que, cuando no se hace uso de paréntesis para separar unos operadores, de otros, las únicas secuencias válidas son:

```
. AND . . NOT .
. OR . . NOT .
. EQV . . NOT .
. NEQV . . NOT .
```

No puede emplearse la secuencia . NOT . . NOT . , que es inválida.

5.3. EXPRESIONES LOGICAS

Hasta aquí, hemos hablado, y empleado en los ejemplos, del concepto *expresión lógica* FORTRAN. Vamos a ocuparnos ahora de definirlo.

Una expresión lógica se emplea en FORTRAN para expresar un determinado cálculo lógico o booleano. Cuando se evalúa una expresión lógica se produce un resultado también de tipo lógico, que como sabemos, es únicamente cierto o falso, . TRUE . o . FALSE . , respectivamente.

Una expresión lógica en FORTRAN puede estar constituida por los ocho elementos que se indican a continuación:

1. Un única constante lógica.

   . TRUE . o . FALSE .

2. Una variable definida en el programa como lógica mediante la sentencia de especificación LOGICAL.
3. Un elemento de un conjunto cuyos elementos son todos variables lógicas, por ejemplo, KLO (5)
4. La llamada a una función FORTRAN lógica. Este concepto será estudiado en otro capítulo.
5. Una expresión de relación, como por ejemplo,

   A . EQ . B

6. Los operadores lógicos

   . AND .
   . OR .
   . EQV .
   . NEQV .

   colocados entre cualquiera de las ocho formas que completan la relación que estamos enunciado. Por ejemplo,

   KLO . AND . A . EQ . B (3) . NEQV . C

7. El operador lógico . NOT . precediendo a las otras siete formas restantes de esta lista. Por ejemplo, . NOT . B(3) .
8. Cualquiera de las ocho formas que constituyen esta relación encerradas entre paréntesis, como por ejemplo

   (KLO . EQV . A(3)) . OR . (C(4) . OR . D)

   A continuación se dan algunos ejemplos de expresiones lógicas válidas e inválidas. Supongamos las variables siguientes, cuyos tipos se indican:

E, VARI	Reales
I, J, K, M	Enteras
L, R	Lógicas
CAR1, CAR2	Carácter
C, N	Complejas

Entonces será:

(VARI * I . GT . K) . AND . L	Válida.
R . AND . . NOT . (J. GT . M)	Válida.
E . AND . R	Inválida. La variable E no se ha definido como lógica
. NOT . R	Válida.
(E + 5.0E3 . LT . 3 * E) . AND . . NOT. R	Válida.
(CAR1 . EQ . CAR 2) . NE. L	Válida.
. AND . (I. EQ . J)	Inválida. Falta un primer elemento delante del operador . AND .
E . EQ . (2 * C)	Válida.
(5.OE + 2 . LT . N) . AND . R	Inválida. Una variable compleja no puede acompañar al operador . LT .
(CAR 2 . EQ . 'NOMBRE') . AND . . NOT . R	Válida.

5.4. PRIORIDAD DE OPERACIONES AL EMPLEAR EXPRESIONES LOGICAS

En el párrafo 2.2, dedicado a las expresiones aritméticas, nos referimos ya a la formación de expresiones aritméricas FORTRAN y cómo el resultado o evaluación de las mismas estaba sometido a unas prioridades (véase regla 3ª de ese párrafo) para el caso de no emplear paréntesis.

Al introducir las expresiones y operaciones lógicas en los cálculos FORTRAN, es necesario que ampliemos aquellas reglas que, con lo dicho hasta aquí, quedan reflejadas en la relación siguiente:

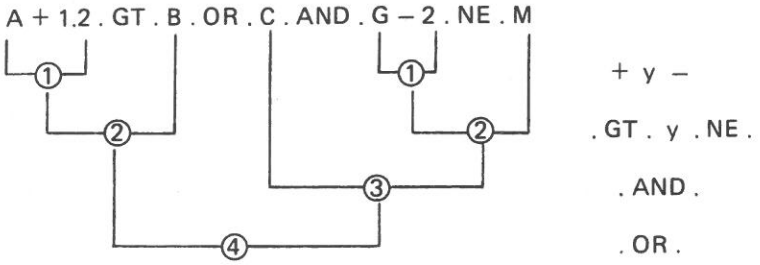
	OPERACION	PRIORIDAD
OPERACIONES ARITMETICAS	Cálculo de potencias (**)	1
	Multiplicación y división (*, /)	2
	Suma y resta (+, −)	3
OPERADORES DE RELACION	Operadores de relación GT ., . GE ., . LT ., . LE ., . EQ ., . NE .	4
OPERADORES LOGICOS	Operadores lógicos:	
	. NOT .	5
	. AND .	6
	. OR .	7
	. EQV . y . NEQV .	8

Cuando en una expresión se dan jerarquias iguales o del mismo nivel, éstas se ejecutan de izquierda a derecha, en general, excepto para el caso de las potencias que, como sabemos, se fectuan de derecha a izquierda.

Veamos, por ejemplo, cómo se realizarían las operaciones indicadas en la expresión siguiente:

A + 1.0 . GT . B . OR . C . AND . G − 2 . NE . M

Como ya hicimos en otra ocasión, los números encerrados en círculos indican el orden de ejecución y los símbolos al margen, la propia operación

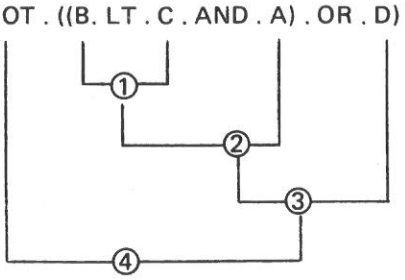


La norma general a seguir es la de hacer uso de paréntesis, con el fin de aumentar la claridad de ejecución de las expresiones disminuyendo al mismo tiempo el riesgo de error. Como en las expresiones aritméticas, cuando se tienen expresiones lógicas, se evaluan en primer lugar los contenidos de los paréntesis.

Por ejemplo, en la expresión

```
. NOT .. ((B . LT . C . AND . A) . OR . D)
```

el orden de ejecución sería:



. LT . } 1er Paréntesis
. AND . }

. OR . 2º Paréntesis

. NOT .

5.5. SENTENCIA DE ASIGNACION LOGICA

En el Capítulo 2, al hablar de sentencias de asignación (2.7), enunciamos que existían tres categorías: asignación aritmética, asignación lógica y asignación carácter. En aquella ocasión estudiamos la primera clase; vamos ahora a tratar de la segunda, dejando para más adelante la última de ellas.

Las sentencias de *asignación lógica* sirven para ordenar al calculador el almacenar en una variable lógica el resultado obtenido (cierto o falso) una vez evaluada una expresión lógica. Es decir, su forma general es

```
vl = el
```

siendo: v_l el nombre de una variable, con o sin subíndice, definida en el programa como una variable tipo lógica (LOGICAL)

e_l una expresión lógica FORTRAN válida.

Una vez más recordamos aquí que el significado del signo "igual" es el de "almacenar en v_l el valor encontrado para e_l".

Si G es una variable definida en el programa como lógica, H un conjunto lógico de cinco elementos y de una sola dimensión, los que siguen son ejemplos de sentencias de asignación lógica:

```
H(2) = . TRUE .
G = . TRUE .
H(5) = . NOT . G
G = H(2) . OR . H(5)
```

Mediante la primera sentencia, H(2) toma un valor *cierto;* G será también *cierta,* en el segundo ejemplo; en el tercer caso, a H(5) se le asignará un valor *falso;* con la última sentencia G tomará el valor *verdadero.* Todo ello supone que estas sentencias se hubieran ejecutado en secuencia y que las variables G y H se hubieran definido como lógicas mediante unas sentencias, como por ejemplo,

```
DIMENSION H(5)
LOGICAL G, H
```

Obsérvese el empleo de la sentencia DIMENSION para definir datos lógicos que forman un conjunto.

5.6. SENTENCIA IF LOGICO

Ya se ha estudiado anteriormente el IF aritmético (2.11). Pero en FORTRAN 77 existen tres clases de sentencias IF:

```
IF aritmético
IF lógico
Bloque IF
```

Estudiaremos ahora el segundo de ellos. El IF lógico permite el uso de las expresiones lógicas, anteriormente estudiadas, para poder efectuar bifurcaciones dentro de un programa FORTRAN.

Su forma general o sintaxis es la siguiente:

```
IF (el) s
```

siendo: e_l una expresión lógica FORTRAN

s una sentencia FORTRAN ejecutable cualquiera, excepto un DO, otro IF lógico, una END o un *bloque* IF con sus sentencias anejas (ELSE IF, ELSE, END IF o THEN).

El significado de esta sentencia es el siguiente: se evalúa, en primer lugar, la expresión lógica e_l; si su valor es *cierto* se ejecuta la sentencia s y luego la siguiente en secuencia al IF; si su valor es *falso*, pasa a ejecutarse directamente la siguiente instrucción en secuencia no teniendo lugar la ejecución de s. Esta semántica queda recogida en el organigrama que se indica en la figura 5.1.

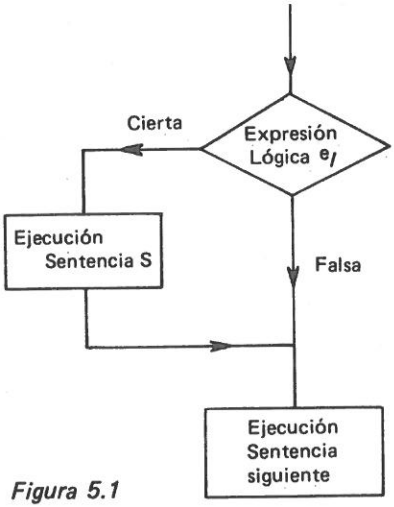


Figura 5.1

Por ejemplo, el siguiente par de sentencias pertenecientes a un programa, tales como

```
IF (A. AND. B) X = X + 4
Y = X * 2 + C
```

ordenarían al sistema de cálculo evaluar la expresión lógica

```
A . AND . B
```

de manera que si esta fuera *cierta* se ejecutaría la sentencia aritmética

```
X = X + 4
```

en primer lugar y luego la Y = X * 2 + C. Ahora bien; si la expresión lógica fuera *falsa*, entonces se ejecutaría directamente

```
Y = X * 2 + C
```

Otros ejemplos válidos de esta misma sentencia, serían:

```
IF (A . GT . (B + C)) WRITE (1,100) A
IF (KM . EQ . 0) GO TO 25
IF (C . NEQV . D) READ (1,200) A, B, C
```

5.7. BLOQUE IF

El conjunto de sentencias conocido en FORTRAN 77 con el nombre de *bloque* IF, permite comprobar, en primer lugar, si una condición es cierta o falsa para en cualquier caso ejecutar un grupo u otro de sentencias, finalizando el proceso en un punto o sentencia común. Este es el *bloque* IF utilizado en el caso de poseer una *doble alternativa*. El organigrama sería el indicado en la figura 5.2.

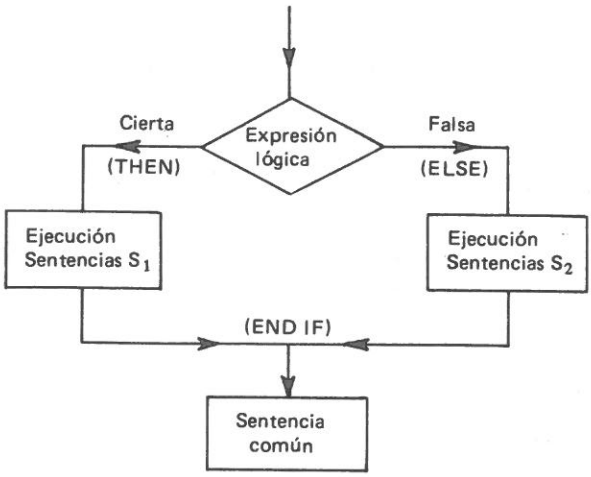
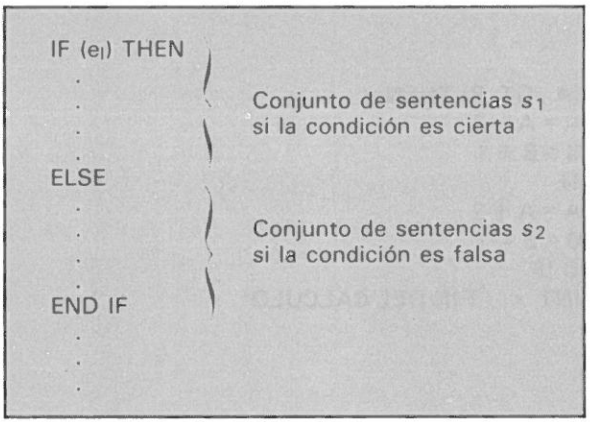


Figura 5.2

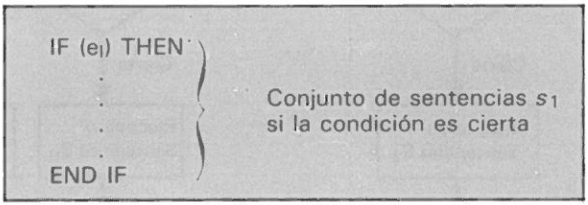
La sintaxis FORTRAN es la siguiente:



siendo e_l, como en el caso del IF lógico, una expresión lógica FORTRAN.

El significado de este bloque es: si e_l es cierta, entonces (THEN) ejecutar el conjunto de instrucciones que componen las líneas s_1; en caso contrario (ELSE) ejecutar las líneas s_2, para en ambos casos y al finalizar uno u otro camino continuar con la sentencia siguiente a la END IF.

En el *bloque* IF puede que no exista el conjunto de sentencias s_2 en cuyo caso sólamente se ejecuta el conjunto s_1 y ello únicamente cuando la expresión lógica e_l es cierta. En tal situación se omitiría la parte ELSE, quedando la sintaxis reducida a



Obsérvese que en el *bloque* IF reducido no se ha omitido sin embargo la sentencia final END IF. Otra regla a tener en cuenta es que dentro de un *bloque* IF no puede existir nada más que una sentencia ELSE.

Un ejemplo de código empleando el *bloque* IF, sería:

```
      .
      .
      .
IF (A . GT .B) THEN
   A = A – 2
   B = B + 1
ELSE
   A = A + 2
   B = B – 1
END IF
PRINT * ,'FIN DEL CALCULO'
      .
      .
      .
```

mediante el cual se disminuye A en dos unidades y se aumenta B en una, siempre que el contenido de A sea mayor que el contenido de B.

Pero si A es igual o menor que B, entonces A se aumenta en dos unidades mientras que el contenido de B se disminuye en una sola. En ambos casos y al final del proceso, se imprime el mensaje FIN DEL CALCULO.

Dentro de un *bloque* IF puede existir a su vez otro o varios bloques IF ya sea después de la sentencia THEN o después de la sentencia ELSE o bien después de una y otra. Este sería el caso a emplear cuando las *alternativas* son *varias* y no únicamente dos como hemos estudiado hasta aquí. En tal situación el organigrama sería el representado en la figura 5.3.

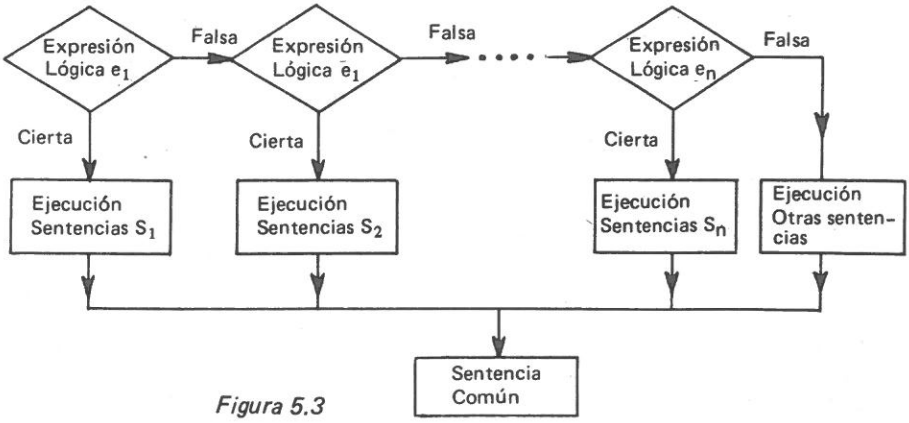
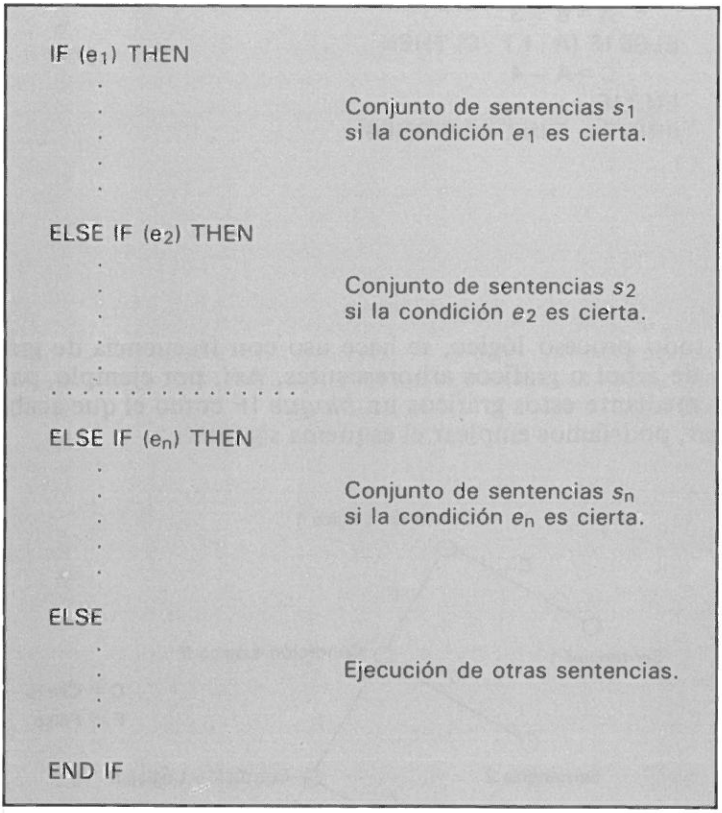


Figura 5.3

El esquema de sintaxis FORTRAN sería ahora así:



En esta representación aparece una nueva sentencia FORTRAN a utilizar en los *bloques* IF: es la sentencia ELSE IF de la que ya habíamos hablado al definir el IF lógico (5.6).

Como ejemplo de codificación FORTRAN y de aplicación de esta estructura podríamos escribir la siguiente:

```
      .
      .
      .
IF (A . GT . B) THEN
      A = B + 3
ELSE IF (A . LT . C) THEN
      C = A – 4
END IF
PRINT *, 'FIN DECISIONES'
      .
      .
      .
      .
```

En todo proceso lógico, se hace uso con frecuencia de gráficos en forma de árbol o gráficos arborescentes. Así, por ejemplo, para representar mediante estos gráficos un *bloque* IF como el que acabamos de estudiar, podríamos emplear el esquema siguiente:

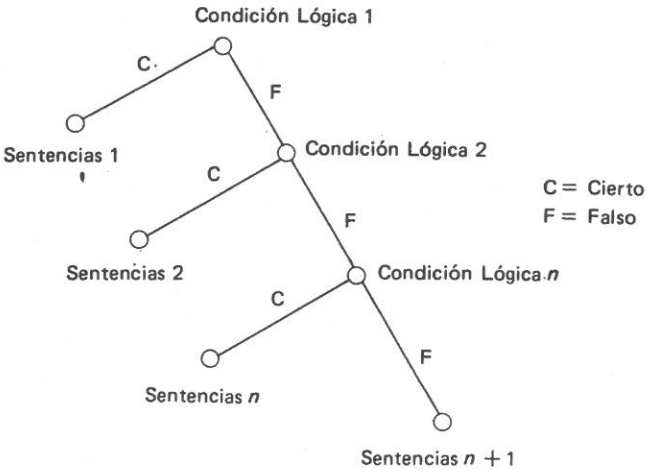


Figura 5.4

Si el esquema lógico arborescente fuera ahora el indicado en la figura 5.5, por ejemplo

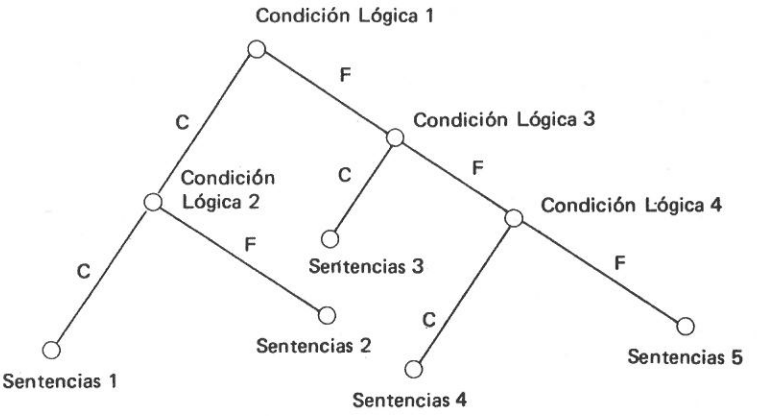


Figura 5.5

entonces la codificación FORTRAN equivalente sería, empleando el *bloque* IF, la siguiente:

```
IF (e1) THEN
   IF (e2) THEN
      Sentencias 1
   ELSE
      Sentencias 2
   END IF
ELSE IF (e3) THEN
      Sentencias 3
   ELSE IF (e4) THEN
         Sentencias 4
      ELSE
         Sentencias 5
END IF
```

Bloque IF interno

Bloque IF más externo

Programa 9. Se trata de escribir un programa FORTRAN para resolver las ecuaciones de segundo grado de la forma

$$Ax^2 + Bx + C = 0$$

de las cuales se conocen sus coeficientes A, B y C que pueden ser números reales cualesquiera. Los coeficientes se leen a través del terminal de pantalla-teclado.

Pueden ocurrir varios casos:

1. Que los coeficientes sean A = B = C = 0, en cuyo caso la ecuación es indeterminada. Escribir el mensaje: ECUACION INDETERMINADA.

2. Que sea A = B = 0 y C ≠ 0. Escribir el mensaje: ECUACION IMPOSIBLE.

3. Que sea A = 0 y B ≠ 0, C ≠ 0 en cuyo caso se trata de un ecuación de primer grado. Escribir su única raíz precedida del mensaje: RAIZ UNICA =.

4. Que todos sus coeficientes sean no nulos, en cuyo caso y dependiendo del valor del discriminante, $B^2 - 4AC$, existirán dos raíces distintas, dos raíces iguales (raíz doble), dos raíces complejas conjugadas. Escribir respectivamente: RAICES REALES, RAIZ DOBLE, RAICES COMPLEJAS y luego sus valores respectivos.

En este programa se hará uso de funciones FORTRAN especiales disponibles para el programador: la función ABS que calcula el valor absoluto de una expresión encerrada entre paréntesis y la función SQRT que calcula la raíz cuadrada también de una expresión. Funciones como éstas existen varias y todas ellas serán estudiadas más adelante. El nombre SQRT viene del Inglés Square Root (raíz cuadrada).

El organigrama para este ejemplo podría ser el representado en la figura 5.6.

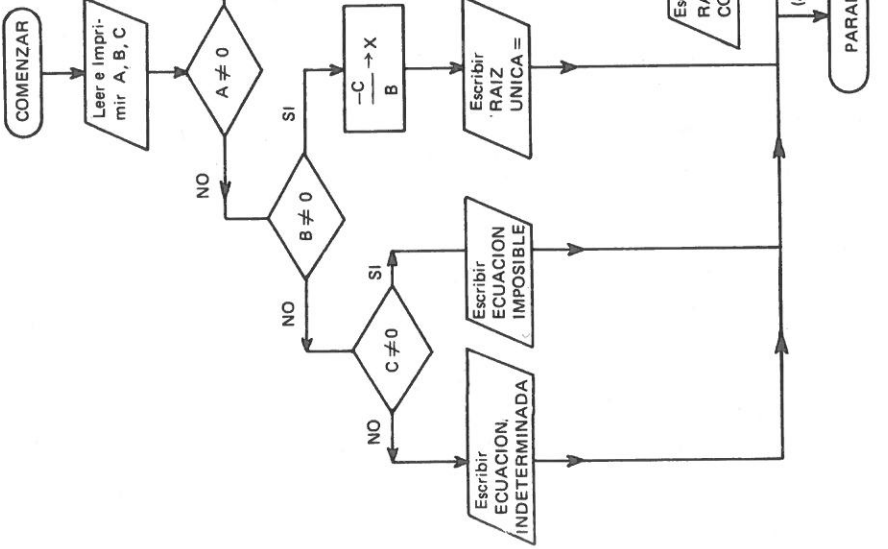

COMENZAR
Leer e Imprimir A, B, C
A ≠ 0
SI
NO
B² – 4AC → DELTA
–B/2A → U
√|DELTA| / 2A → V
DELTA ?
< 0
= 0
> 0
(1)
(2)
(3)
Escribir RAICES COMPLEJAS
Escribir RAIZ DOBLE
Escribir RAICES REALES
B ≠ 0
–C/B → X
Escribir RAIZ UNICA =
C ≠ 0
Escribir ECUACION IMPOSIBLE
Escribir ECUACION INDETERMINADA
(4)
PARAR

Figura 5.6

y la codificación FORTRAN la indicada en la figura 5.7.

```
      PROGRAM ECUA2
C
      READ *,A,B,C
      PRINT *,'COEFICIENTES ',A,B,C
      IF(A.NE.0.0)THEN
         DELTA=B**2-4.0*A*C
         U=-B/(2.0*A)
         V=SQRT(ABS(DELTA))/(2.0*A)
         IF(DELTA)1,2,3
    1     PRINT *,'RAICES COMPLEJAS:',
     1    'PARTE REAL=',U,'PARTE IMAGINARIA=',V
         GO TO 4
    2     PRINT *,'RAIZ DOBLE=',U
         GO TO 4
    3     PRINT *,'RAICES REALES:',U+V,U-V
      ELSE IF(B.NE.0.0)THEN
         PRINT *,'RAIZ UNICA=',-C/B
         ELSE IF(C.NE.0.0)THEN
            PRINT *,'ECUACION IMPOSIBLE'
         ELSE
            PRINT *,'ECUACION INDETERMINADA'
      END IF
    4 STOP
      END
```

Figura 5.7

EJERCICIOS RESUELTOS

5.1. Utilizando signos matemáticos, como $<, >, \leqslant$, etc., indicar cuál es el significado de las siguientes expresiones lógicas FORTRAN

a) J . GE . K
b) (B + 3.8) . NE . (A + X)
c) (VELO ** 2) . LE . (5 + A)
d) 501.2 . EQ . (A + 3.0 * C)
e) A . GE . B
f) (A ** 3) . LT . (C ** 2)
g) K . GT . (H + 3.2)
h) (3 * Y + Z) . GE . (A + B/7.0)
i) (1 . LE . X) . AND . (X . LE . 2)

Respuesta

a) $J \geqslant K$
b) $(B + 3.8) \neq A + X$
c) $VELO^2 < (5 + A)$
d) $501.2 = A + 3C$
e) $A \geqslant B$
f) $A^3 < C^2$
g) $K > (H + 3.2)$
h) $(3Y + Z) \geqslant (A + B/7)$
i) $1 \leqslant X \leqslant 2$

5.2. Suponiendo que las siguientes variables toman los valores que se indican

```
I = 3
J = 8
A = 10.0
B = 40.0
```

deducir los valores lógicos, cierto o falso, que tomarían las expresiones lógicas FORTRAN siguientes:

a) I . GE . J
b) A ** 2 . NE . B
c) (I + A * 2) . GT . 2 * B
d) A + B . LT . (B * 3)
e) J . EQ . I
f) B/A . LE . (B–A)

Respuesta

a) Falso
b) Cierto
c) Falso
d) Cierto
e) Falso
f) Cierto

5.3. Dadas las variables con los valores que se indican

```
A = 2      D = 2.5     J = 3
B = 5      E = –4      K = –2
C = 10     I = 2       F = .FALSE.
                       T = .TRUE.
```

Deducir el valor lógico de cada una de las expresiones lógicas siguientes:

a) T . AND . F . OR . . FALSE
b) A ** I + B . LE . B/C + D
c) I/J . EQ . 2 + K . AND . B/C + D . GE . E + C/D – A ** J
d) C ** 2 + D * E – 10.0E+2 . LE . B + 2.0 * C . AND . (. FALSE . . OR . . NOT . . FALSE .)
e) (B * J + 3.0) . EQ . (D–E) . AND . (. NOT . F)

Respuesta

a) Cierta
b) Falsa
c) Cierta
d) Cierta
e) Falsa

5.4. Suponiendo que las variables lógicas X, Y, Z tienen los valores

X = .FALSE.
Y = .FALSE.
X = .TRUE.

encontrar el valor de cada una de las expresiones lógicas siguientes:

a) X .NEQV. Y
b) X .OR. .NOT. Z
c) X .EQV. (.NOT. X)
d) X .OR. (Y .AND. Z)
e) X .OR. Z .AND. Y .OR. Z

Respuesta

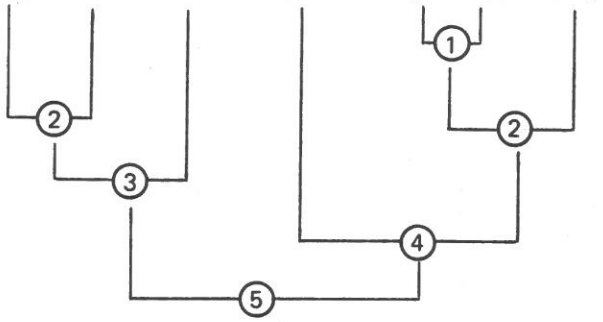
a) Falsa
b) Falsa
c) Falsa
d) Falsa
e) Cierta

5.5. Deducir el orden en que se realizan las siguientes expresiones lógicas:

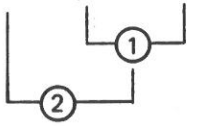
a) ((2.5 + A) .LT. B) .OR. (C .AND. ((5 * G) .NE. Z))
b) .NOT. (X .OR. Y)
c) .NOT. X .OR. Y
d) (.NOT. X) .OR. Y

Respuesta

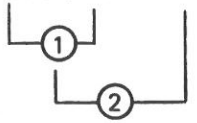
a) ((2.5 + A) .LT. B) .OR. (C .AND. ((5 * G) .NE. Z))



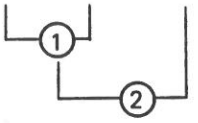
b) . NOT . (X . OR .Y)



c) . NOT . X . OR . Y



d) (. NOT . X) . OR . Y



5.6. Dada la codificación siguiente que hace uso del IF lógico

```
      IF (A . GT . B) GO TO 10
         X = A
         B = B + 2
10    X = A + B
```

sustituir esta estructura por un *bloque* IF equivalente que produzca el mismo resultado.

Respuesta

```
IF (A . LE . B) THEN
    X = A
    B = B + 2
END IF
X = A + B
```

5.7. Las variables A, B, y C tienen los valores respectivos

A = 2.0 , B = 3.0 , C = 8.0

Encontrar los valores finales que toman esas variables después de ejecutarse las estructuras siguientes:

a)
```
IF (A . LT . 20.0) THEN
   A = A + 2
   B = B –3
END IF
A = 2 * B
B = 3 * A
```

b)
```
IF (C. GT. A) THEN
      C = 2 * C
      A = B – C
   ELSE IF (C . EQ . 3) THEN
      C = C – 2
   ELSE
      C = C – 5
END IF
C = C + A + B
```

c)
```
IF (C . EQ . B) THEN
      C = A + B
   ELSE IF (C . EQ . A) THEN
      A = B + C
   ELSE
      A = B – C
END IF
B = A + C
```

d)
```
IF (A . NE . C) THEN
      A = 2 * B
   ELSE IF (A . NE . B) THEN
      B = 3 * C
   ELSE
      C = 2 + B
END IF
```

Respuesta

a) A = 0.0 " B = 12.0 " C = 12.0
b) A = –13.0 " B = 3.0 " C = 6.0
c) A = –5.0 " B = 3.0 " C = 8.0
d) A = 6.0 " B = 3.0 " C = 8.0

6

Otras sentencias FORTRAN

Admiro con frecuencia el procedimiento místico de Pitágoras y el mágico secreto de los números

SIR THOMAS BROWNE *(1605-1682)*

6.1. OTRAS SENTENCIAS DE TIPO: INTEGER, REAL E IMPLICIT

En el Capítulo 1 ya se estudiaron algunas sentencias de tipo como las

```
LOGICAL
DOUBLE PRECISION
CHARACTER
```

mediante las cuales podía alterarse el tipo implícito de una variable que por el hecho de comenzar o no su nombre por unas letras (I, J, K, L, M o N) adquiría una clasificación de entera o de real.

Pues bien; este tipo implícito puede alterarse mediante el empleo de las sentencias no ejecutables INTEGER y REAL que pertenecen tanto al FORTRAN básico como al completo y que se conocen con el nombre de *sentencias de tipo* o *sentencias de declaración.*

Las variables enteras pueden declarse explícitamente en un programa mediante el uso de la sentencia INTEGER, en cuyo caso, cualquiera que sea la primera letra de la variable, esta adquirirá el carácter de entera.

Por ejemplo, la sentencia de un programa

```
INTEGER BETA
```

informa al compilador que la variable BETA, que implícitamente es real por ser una B su primer carácter alfabético, queda convertida en entera.

Esta sentencia puede aplicarse, simultáneamente, a varias variables reales para que puedan alojar datos enteros. Por ejemplo

```
INTEGER BETA, A, X, Y
```

Cuando esta sentencia se aplica al nombre de un conjunto que implícitamente es real, todos sus elementos quedan convertidos en enteros. Así por ejemplo, el conjunto BETA de cuatro elementos y de una sola dimensión podría convertirse en entero así:

```
INTEGER BETA (1), BETA (2), BETA (3), BETA (4)
```

pero ese efecto puede lograrse de una forma más abreviada empleando la sentencia

```
INTEGER BETA (4)
```

es decir, esta sentencia de tipo podría haberse desglosado en las

```
DIMENSION BETA (4)
INTEGER BETA
```

y ambas formas son equivalentes. De la primera forma se deduce que esta sentencia puede contener información sobre las dimensiones y número de elementos de un conjunto. Por ello y recordando lo estudiado en 2.5 al hablar de declaración de conjuntos, otros ejemplos de esta sentencia podrían ser:

```
INTEGER PIEREC (1:3, 1:4)
INTEGER A(2:10), B(0:30), C(1:10, 1:15)
```

utilizando límites inferior y superior para definir el número total de elementos correspondiente a cada dimensión del conjunto.

Según todo lo indicado, la forma general de esta sentencia sería

```
INTEGER nombre 1 (dimensión 1), nombre 2 (dimensión 2), ...
```

siendo: *nombre i*, el nombre de una variable, el nombre de un conjunto o el nombre de una función (este concepto será estudiado en otro capítulo).

dimensión i, el indicador de las dimensiones de *nombre i*, caso de tratarse de un conjunto. Este concepto es opcional.

Una variable que implícitamente sea entera, puede convertirse en real empleando la sentencia de tipo o declaración REAL. Por ejemplo

```
REAL  KM
```

significa que el compilador deberá considerar la variable KM, implícitamente entera, como real.

Las mismas observaciones hechas anteriormente sobre el uso de la sentencia INTEGER pueden repetirse para la REAL, en cuanto a significado y empleo en ella de conjuntos y dimensiones.

La forma general de una sentencia REAL, será:

```
REAL nombre 1 (dimensión 1), nombre 2 (dimensión 2), ...
```

Recuérdese que, como en el caso de INTEGER, el concepto *dimension i* puede faltar.

Por último, vamos a estudiar la sentencia de tipo, IMPLICIT. Esta sentencia se utiliza en un programa para cambiar o confirmar el carácter implícito de una variable, de un conjunto de variables, de una función externa o de una función de sentencia, conceptos, estos dos últimos, a estudiar más adelante.

Supongamos que dentro de un programa tenemos un gran número de variables que van a ser enteras y que por razones de significado no interesa que comiencen por los caracteres I, J, K, L, M o N, sino por los caracteres D, E, F, G, H, I, J. Una forma de comunicar al compilador este nuevo carácter entero de todas ellas, sería escribir una sentencia tipo INTEGER incluyendo en ella la lista de las variables cuyo tipo quiere indicarse explícitamente. Pero otra forma mucho más abreviada de hacerlo sería escribiendo

```
IMPLICIT INTEGER (D-J)
```

lo cual indicaría que todas aquellas variables del programa que comiencen por los caracteres D, E, F, G, H, I, o J son enteras.

De la misma forma

```
IMPLICIT REAL (I, K-N)
```

significaría que, en un programa, todas aquellas variables que comiencen por la letra I serán consideradas como reales. Lo mismo con aquellas que comiencen por K, L, M o N.

Otro ejemplo:

```
IMPLICIT CHARACTER*4 (A)
```

significa que todas las variables que comiencen por la letra A serán consideradas como *carácter* y con una longitud de cuatro caracteres cada una.

En una misma sentencia IMPLICIT se pueden definir varios tipos. Por ejemplo

```
IMPLICIT REAL (A-D, U-Z), LOGICAL (L), INTEGER (E-G, J)
```

Según todo lo dicho, la forma general de esta sentencia es

```
IMPLICIT tipo1 (a1), tipo2 (a2),.....
```

siendo: *tipoi*, uno de los tipos INTEGER, REAL, DOUBLE PRECISION, LOGICAL, CHARACTER o COMPLEX. Este último será estudiado a continuación. En el FORTRAN básico únicamente INTEGER, REAL, LOGICAL o CHARACTER.

ai, es una letra única o una serie de ellas, indicándose la primera y la última en orden alfabético, separadas por un guión (signo menos).

6.2. CONSTANTES Y VARIABLES COMPLEJAS

Al hablar en 1.3. de constantes y variables FORTRAN nos referimos, por primera vez, a las constantes y variables complejas como un elemento más susceptible de ser empleado cuando se trabaja con el FORTRAN completo. Vamos ahora a definirlas y a conocer como se hace uso de ellas (suponemos que el lector posee ya un conocimiento previo de la aritmética y álgebra de los números complejos).

Un *dato complejo* es una aproximación al valor de un número complejo y concretamente una *constante compleja* FORTRAN es un par ordenado de constantes enteras o reales FORTRAN separadas por una coma y encerradas entre paréntesis. La primera de ellas representa la

parte real del número complejo y la segunda su parte imaginaria. He aquí algunos ejemplos de constantes complejas en los que es $i^2 = -1$.

CONSTANTE FORTRAN	SIGNIFICADO
(36.0, –2.05)	36 –2,05 i
(–0.6E2, 7.3E–1)	–60 + 0,73 i
(5.E1, –8.E–1)	50 – 0,8 i
(6, –28)	6 – 28 i
(8.2, 4.)	8,2 + 4 i
(5.1E2, 3.)	510 + 3 i

Algunos compiladores, aunque esto no sea standard, admiten el empleo de constantes doble precisión para formar las constantes complejas. En este caso también serían válidas las constantes complejas siguientes:

```
(5.2 D+2, –4.8
(6., –33.7D–1)
(6.2E+1, 7.3D+2)
```

Toda constante compleja ocupa dos unidades de memoria consecutivas.

Mediante las *variables complejas* se denominan las constantes complejas o bien posiciones de la memoria ocupadas por datos complejos. Para definir una variable compleja ha de emplearse la sentencia de tipo COMPLEX. Por ejemplo, la sentencia

```
COMPLEX BAR, IMAG
```

define al compilador dos variables como complejas: la BAR y la IMAG.

En esta sentencia pueden también definirse conjuntos formados por elementos todos ellos datos complejos FORTRAN. Así

```
COMPLEX BAR(1:5, 2:8)
```

define un conjunto de dos dimenciones, de datos complejos, conteniendo información (límites inferior y superior) sobre los subíndices.

Como resumen, la forma general de esta sentencia es

```
COMPLEX nombre 1 (dimensión 1), nombre 2 (dimensión 2), ...
```

siendo los significados de *nombre i* y *dimensión i* los mismos estudiados anteriormente para INTEGER y REAL.

6.3. EXPRESIONES COMPLEJAS

Definidas las constantes y variables complejas podemos ahora definir lo que es una *expresión compleja*. Se define como tal una expresión aritmética cuyos valores son del tipo complejo. Por ejemplo si A y B estuvieran definidas como variables complejas,

2 * A	doble de una variable compleja A
3.5 + B	equivalente a sumar 3,5 + 0i con B
5 * (4.3, –2.1)	equivalente a multiplicar por 5 la constante compleja 4,3–2, 1i

son expresiones complejas

Los cuadros 2.2 y 2.3 contenían las reglas de combinación de dos datos cualesquiera enteros, reales o doble precisión. Vamos ahora a ampliar esta información haciendo intervenir los datos complejos en el cálculo de una expresión.

El cuadro 6.1, indica cómo es el resultado obtenido al combinar dos datos cualesquiera utilizando para ello los operadores +, –, * y /.

+, –, *, /	DATO ENTERO	DATO REAL	DATO D. PRECISION	DATO COMPLEJO
DATO ENTERO	RESULTADO ENTERO	RESULTADO REAL	RESULTADO D. PRECISION	RESULTADO COMPLEJO
DATO REAL	RESULTADO REAL	RESULTADO REAL	RESULTADO D. PRECISION	RESULTADO COMPLEJO
DATO D. PRECISION	RESULTADO D. PRECISION	RESULTADO D. PRECISION	RESULTADO D. PRECISION	NO PERMITIDO
DATO COMPLEJO	RESULTADO COMPLEJO	RESULTADO COMPLEJO	NO PERMITIDO	RESULTADO COMPLEJO

Cuadro 6.1

Si los datos se combinan empleando la exponenciación, **, entonces los resultados y las reglas de combinación están contenidos en el cuadro 6.2

	**	EXPONENTE			
		DATO ENTERO	DATO REAL	DATO D. PRECISION	DATO COMPLEJO
BASE	DATO ENTERO	RESULTADO ENTERO	RESULTADO REAL	RESULTADO D. PRECISION	RESULTADO COMPLEJO
	DATO REAL	RESULTADO REAL	RESULTADO REAL	RESULTADO D. PRECISION	RESULTADO COMPLEJO
	DATO D. PRECISION	RESULTADO D. PRECISION	RESULTADO D. PRECISION	RESULTADO D. PRECISION	NO NO PERMITIDO
	DATO COMPLEJO	RESULTADO COMPLEJO	RESULTADO COMPLEJO	NO PERMITIDO	RESULTADO COMPLEJO

Cuadro 6.2

Como es lógico el cuadro 6.1 queda sensiblemente reducido cuando se utiliza un FORTRAN correspondiente al standard básico. Como en este caso únicamente se emplean datos enteros o reales, el cuadro equivalente es el 6.3

+, −, *, /	DATO ENTERO	DATO REAL
DATO ENTERO	RESULTADO ENTERO	RESULTADO REAL
DATO REAL	RESULTADO REAL	RESULTADO REAL

Cuadro 6.3

Para el caso de la exponenciación, el cuadro 6.2 se convierte y reduce en el cuadro 6.4.

Analicemos algunso casos para ver como se llega a los resultados que se indican en los cuadros. En el cuadro 6.1 se dice que la combinación entero-real, produce un resultado R real. Para ello, primeramente el

**		EXPONENTE	
		DATO ENTERO	DATO REAL
BASE	DATO ENTERO	RESULTADO ENTERO	RESULTADO REAL
	DATO REAL	RESULTADO REAL	RESULTADO REAL

Cuadro 6.4

dato entero E_1 es convertido en real y luego sumado con el segundo dato real R_2, es decir:

$$R = REAL\ (E_1) + R_2$$

El resultado real-doble precisión es doble precisión DP. Ello se logra convirtiendo el dato real R_1 en doble precisión y luego sumándolo al dato doble precisión DP_2 es decir:

$$DP = DOBLE\ \ PRE(R_1) + DP_2$$

La combinación real-complejo produce un resultado complejo C. Para ello, el dato real R_1, se convierte en un número complejo de la forma (R_1, 0.) y luego se suma con el dato complejo C_2, es decir:

$$C = COMPLEJO\ (R_1, 0.) + C_2$$

Tratándose, por ejemplo del cuadro 6.2, la combinación entero-complejo produce un resultado complejo C y ello de la siguiente forma:

$$C = COMPLEJO\ (REAL(E_1), 0.) ** C_2$$

y la combinación real-doble precisión, se efectúa así:

$$DP = DOBLE\ \ PRE(R_1) ** DP_2$$

El programador debe conocer estos detalles únicamente para saber en todo momento como opera la máquina. No obstante, esta operativa se lleva a cabo automáticamente.

En el Capítulo 2, párrafo 2.7, hemos hablado ya de las sentencias de asignación aritméticas. Es el momento ahora de ampliar un poco todo lo dicho allí, haciendo intervenir los conceptos sobre datos complejos que acabamos de estudiar.

En toda sentencia de asignación aritmética intervienen dos partes: *v* y *e*. Anteriormente se dijo que *v* era el nombre de una variable con o sin subíndice, de tipo entera, real o doble precisión. Ahora añadiremos que además puede ser una variable con o sin subíndice, definida o declarada como compleja (sólo en el FORTRAN completo).

También dijimos que *e* era una expresión aritmética FORTRAN; también podría ser una expresión aritmética pero compleja.

Con esta ampliación obtendremos como resultado la *asignación compleja* a incluirla dentro de la asignación aritmética general

Al utilizar los datos complejos, el cuadro 2.4 se transformaría ahora en el cuadro 6.5 que es una ampliación del primero y en el que C significa Complejo.

TIPO DE e / TIPO DE v	E	R	DP	C
E	E	E	E	$E_{(1)}$
R	R	R	R	$R_{(2)}$
DP	DP	DP	DP	$DP_{(3)}$
C	$C_{(4)}$	$C_{(5)}$	$C_{(6)}$	$C_{(7)}$

Cuadro 6.5

(1) La parte real del número complejo se convierte en entera (produciéndose truncamiento o pérdida de la parte fraccionaria, si fuese necesario). La parte imaginaria se pierde.

(2) La parte real del número complejo se asigna a *v*; la parte imaginaria se pierde.

(3) La parte real se convierte en doble precisión y se asigna a *v*; la parte imaginaria se pierde.

(4) Se convierte *e* en real y se asigna como la parte real de *v*, cuya parte imaginaria será cero.

(5) Se asigna *e* directamente a *v* como parte real del número complejo. La parte imaginaria de *v* será cero.

(6) Se asigna *e* como variable real a la parte real de *v*; la parte imaginaria de *v* será cero.

(7) Se asigna *e* directamente a *v*.

Como resumen diremos que la ejecución de una sentencia de asignación aritmética en la que intervienen datos complejos, causa la evaluación de la expresión *e* siguiendo las reglas contenidas en los cuadros 6.1 y 6.2 y asignando luego dicho valor a *v* según las reglas y observaciones contenidas en el cuadro 6.5.

Si A es una variable compleja, la asignación más sencilla haciendo intervenir un dato complejo como por ejemplo 3 + 2,5i, sería:

```
A = (3.0, 2.5)
```

No es válido, sin embargo, asignar a una variable compleja A otras dos reales B, C escribiendo

```
A = (B, C)
```

6.4. OPERACIONES DE E/S DE DATOS COMPLEJOS

Cada dato o valor de tipo complejo se designa por una sola variable en la lista de una sentencia de entrada o salida. Ahora bien; es necesario asignarle dos códigos de formato F, E, D o G en la sentencia FORMAT correspondientes a la de E/S. Esos códigos que pueden ser diferentes, estarán asociados, respectivamente, a la parte real y a la parte imaginaria del dato complejo.

Por ejemplo, supongamos en memoria la variable A que contiene un valor complejo. Para escribir este dato por el terminal pantalla-teclado, sería necesaria una codificación como la siguiente:

```
      COMPLEX A
        .
        .
      WRITE (1,100) A
100   FORMAT (F10.3, 2X, F8.2)
        .
        .
```

Con esta codificación se visualizaría la parte real del número complejo A según el formato F10.3 y la parte imaginaria según el F8.2 y ambas partes estarían separadas por dos espacios en blanco. Podrían haberse empleado también códigos de formato diferentes para A, como F10.3 y E14.8.

Terminaremos esta parte dedicada a los datos complejos añadiendo que el lenguaje FORTRAN suministra el usuario varias rutinas o pro-

cedimientos con los que resolver automáticamente problemas matemáticos del campo complejo. Se estudiarán en el apéndice correspondiente.

6.5. EXPRESIONES TIPO CARACTER

Una vez estudiadas en el Capítulo 1 las constantes y variables *carácter*, vamos ahora a ampliar un poco más estos conceptos, introduciendo lo que podríamos llamar aritmética de los datos tipo *carácter*. Para ello vamos a exponer algunos conceptos pertenecientes a este área, como son:

Conjuntos carácter
Series de caracteres
Operador carácter
Expresión carácter

a) Conjuntos carácter

En FORTRAN no sólo pueden manejarse conjuntos formados por datos enteros, reales, lógicos, etc. También pueden definirse y utilizarse conjuntos formados por datos *carácter* y ello empleando DIMENSION y CHARACTER para declararlos. Por ejemplo, un conjunto denominado CARAT de 20 elementos de manera que cada elemento esté formado por una variable *carácter* de longitud 12, se definiría así:

```
CHARACTER*12 CARAT
DIMENSION CARAT(20)
```

o bien de forma más abreviada

```
CHARACTER*12 CARAT(20)
```

Esas 20 variables *carácter* podrían introducirse en memoria mediante una lectura, es decir, empleando una sentencia READ, por ejemplo, de la forma:

```
      DO 100 I = 1,20
100   READ *, CARAT (I)
```

En este caso cada variable CARAT(I) se leería como ya sabemos de un registro diferente; cada una de las variables *carácter* que se leyera debería estar encerrada entre comillas (’), como por ejemplo:

```
'RAZON'
'UNIDAD'
'SESIONES'
   .
   .
   .
```

Cuando se emplee una sentencia READ con formato para ingresar en memoria datos *carácter,* debemos recordar todo lo expuesto sobre este tema en 4.6 al hablar del código de formato A.

Como veremos más adelante, el valor 'RAZON' también podría haber sido asignado a la variable *carácter* CARAT(1), 'UNIDAD' a la CARAT(2) y así sucesivamente, empleando sentencias de asignación *carácter,* como

```
CARAT(1) = 'RAZON'
```

en lugar de sentencias READ. Ahora bien; este último procedimiento deja en libertad al programador para introducir en memoria datos variables cada vez que se ejecute el programa.

b) Series de caracteres

Ya hemos definido un dato *carácter* como una sucesión de uno o más caracteres FORTRAN. Diremos que una parte contigua de un dato *carácter* forma una *serie de caracteres* (en Inglés, substring) (*). Su tipo es *carácter.* Como veremos en los ejemplos que siguen, una serie de caracteres pueden reconocerse por su *nombre.* Se supone que cada uno de los caracteres que componen una *serie de caracteres* o, por brevedad, una *serie*, ocupan un lugar dentro de ella, o lo que es equivalente: es como si los caracteres de la serie estuvieran numerados, comenzando en el uno y recorriendo la serie de izquierda a derecha. Así, por ejemplo, si consideramos que la variable tipo CHARACTER, denominada SERIE, tuviera el valor

```
'INSCRITO_EN_EL_REGISTRO_GENERAL'
```

una *serie* extraída de la misma podría ser:

```
'EN_EL_REGISTRO'
```

(*) Este concepto también se encuentra en algunos textos con el nombre de *tira de caracteres.*

formada por los elementos del diez al veintitrés del primitivo dato. Entonces, el *nombre* de esta serie sería:

```
SERIE (10:23)
```

De la misma forma

```
SERIE (5:12)
```

equivaldría a

```
'RITO_EN_EL_R'
```

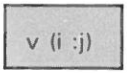
Por último

```
'REGISTRO'
```

de denominaría por

```
SERIE (16:23)
```

En general, el nombre de una serie tiene la forma



siendo: *v* el nombre de una variable o de un elemento de un conjunto de tipo *carácter.*

i, j son expresiones enteras que indican los límites inferior y superior, respectivamente, de la serie tomada del dato *carácter.*

Cuando se omite el límite inferior *i*, el compilador toma el valor 1. Por ejemplo, SERIE (:5) vale

```
'INSCR'
```

Cuando se omite el límite superior *j*, el compilador toma para él la longitud total del dato *carácter.* Así por ejemplo, SERIE(25:) equivale a

```
'GENERAL'
```

Si se omiten ambos límites, *i*, *j*, el compilador toma todo el dato *carácter*. Por ejemplo, SERIE (:) vale

```
'INSCRITO_EN_EL_REGISTRO_GENERAL'
```

Por último, SERIE (3:3) vale 'S' y SERIE (24:24) , '_' (blanco).

c) *Operador carácter*

En FORTRAN 77 completo existe un operador especial para *concatenar* o unir series de caracteres. Este operador está formado por dos barras de la forma

y se denomina *operador de caracteres* u *operador de concatenación*.

Toda operación de concatenación, sobre dos series de caracteres, da como resultado otra serie formada por ambas, una a continuación de la otra, de manera que la longitud de la serie obtenida será la suma de las longitudes de los datos que intervienen.

Por ejemplo: sean las series

```
'INSCRITO_EN_' y 'EL_REGISTRO_2'
```

de longitudes respectivas 12 y 13; entonces al concatenar ambas mediante la operación

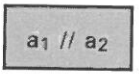
```
'INSCRITO_EN_' // 'EL_REGISTRO_2'
```

se obtendría como resultado la serie

```
'INSCRITO_EN_EL_REGISTRO_2'
```

cuya longitud es de 25 caracteres.

Podremos, pues, expresar la forma general de una operación de concatenación así:



siendo: a_i los dos operandos, series de caracteres, que intervienen en la operación: a_1 es el operando primero o de la izquierda y a_2 el segundo o el de la derecha.

Ya se comprende que el operador de concatenación no puede aplicarse nada más que sobre datos definidos como *carácter.*

d) *Expresión carácter*

Otro elemento del lenguaje FORTRAN 77 es el de expresión *carácter.* Las formas válidas de una expresión *carácter* son las siguientes:

1. Una constante de tipo *carácter:*

```
'SEGUNDO' , 'KILOMETRO_CERO'
```

2. El nombre simbólico de una variable tipo *carácter:*

```
COMP , KILO
```

3. Un elemento de un conjunto *carácter:*

```
SERIE(3) , COLA(2, 3)
```

4. Un serie de caracteres como

```
SERIE(2:8) , COTA(I) (4:10)
```

5. Una llamada a una función especial FORTRAN destinada al manejo automático de caracteres (serán estudiadas más adelante):

```
CHAR(10) , LEN(SERIE)
```

6. La concatenación de dos cualesquiera de los cinco casos anteriores:

```
'SEGUNDO' // COMP , SERIE(3) // COTA(I) (4:10)
```

En FORTRAN básico únicamente pueden emplearse los elementos 1, 2 y 3, ya que no existen ni las *series* ni la *concatenación.*

El empleo sucesivo del elemento 6 en el FORTRAN completo dará lugar a concatenaciones del tipo múltiple, como por ejemplo:

```
'SEGUNDO' // COMP // SERIE(3)
```

es decir, una expresión *carácter* puede contener cualquier número de concatenaciones. En este caso y de forma análoga a como se hacía en las expresiones aritméticas o en las lógicas, es posible el empleo de paréntesis para indicar el orden en que se realizan las concatenaciones, aunque, como veremos a continuación, el resultado que se obtiene no depende realmente del uso del paréntesis.

Por ejemplo

```
(A // B) // C
```

produce el mismo resultado que la expresión

```
A // (B // C)
```

o que la expresión sin paréntesis

```
A // B // C
```

6.6. SENTENCIA DE ASIGNACION CARACTER

Anteriormente hemos definido ya lo que se entiende por asignación aritmética (2.7) y asignación lógica (5.5) de manera que la última sentencia de asignación que vamos a estudiar es la asignación carácter.

La forma general de una sentencia de *asignación carácter* es

```
v = ec
```

siendo: v el nombre de una variable definida como *carácter*, el nombre de un elemento de un conjunto *carácter* o una *serie*. (Este último no utilizable en el FORTRAN básico).

e_c es una expresión *carácter* válida.

La ejecución de una sentencia de asignación carácter da lugar a una evaluación de la expresión e_c asignándose luego el valor obtenido a v.

Ejemplos:

a) `SERIE(2:3) = '24'`

Esta sentencia asigna el valor 24 a los caracteres segundo y tercero de la serie de caracteres denominada SERIE, reemplazando los que ya existieran en esa posición.

b) La codificación

```
CHARACTER*5 A,B
A = 'LISTA'
B = 'SESGO'
B = A
```

asignaría las constantes 'LISTA' y 'SESGO' a las variables A y B, respectivamente, cuyas longitudes son de cinco caracteres. La última sentencia de asignación carácter

```
B = A
```

llevaría a B el valor de A, es decir, 'LISTA'.

c) Supongamos ahora la codificación:

```
CHARACTER*5 A,B,C*10
A = 'LISTA'
B = 'SESGO'
C = A // B
```

Al final de la misma, el contenido de C será

```
'LISTASESGO'
```

Obsérvese como ha sido necesario definir C como variable *carácter* de longitud diez para poder albergar las longitudes de A y de B.

La asignación carácter está sometida a algunas reglas que vamos a exponer:

– Ninguna de las posiciones-carácter de v pueden estar referenciadas en e_c. Por ejemplo, si SERIE es una variable *carácter* de longitud 20, no estarían permitidas las siguientes asignaciones:

```
SERIE = SERIE (1:3)
SERIE = SERIE (2:5) // SERIE(15:)
```

ya que en la primera se hace referencia en su segundo miembro a los caracteres primero, segundo y tercero del primer miembro.

En la segunda sentencia se hace referencia al segundo, tercero, cuarto y quinto y del 15 en adelante, todos ellos del primer miembro.

El siguiente ejemplo, sin embargo, sería válido:

```
SERIE (:10) = SERIE(11:)
```

porque en el segundo miembro no se hace referencia a ningún elemento del primero.

– Las longitudes de v y de e_c pueden ser diferentes. Sean l_v y l_{e_c} dichas longitudes. Entonces pueden suceder dos casos:

  Si es $l_v > l_{e_c}$, entonces el contenido de e_c se asigna a v cubriéndose con blancos a la derecha las posiciones sobrantes, hasta alcanzar l_v.

  Si es $l_v < l_{e_c}$, antes de producirse la asignación, se pierden los caracteres de más a la derecha de e_c para que resulte una constante *carácter* de la misma longitud que v.

Por ejemplo, supongamos la codificación

```
CHARACTER*5 A,B*8
A = 'LISTA'
B = 'SESIONES'
B = A
```

Al final de la misma, el contenido de B sería

```
'LISTA___'
```

Si la codificación fuera

```
CHARACTER*5 A,B*8
A = 'LISTA'
B = 'SESIONES'
A = B
```

el contenido de A, al final, sería:

```
'SESIO'
```

Veamos otro ejemplo. Supongamos que SER1 tiene el valor *carácter*

```
SER1 = 'LA_PIEZA_NUNCA_ES_ REDONDA'
```

Entonces, la asignación

```
SER1 (10:14) = 'BRILLANTE"
```

daría lugar a

```
SER1 = 'LA_PIEZA_BRILL_ES_REDONDA'
```

Mientras que la asignación

```
SER1 (10:14) = 'NO'
```

produciría este otro resultado:

```
SER1 = 'LA_PIEZA_NO____ES_REDONDA'
```

– Cuando *v* sea una *serie*, e_c se asigna únicamente a esa *serie* y no al dato *carácter* de donde ésta provenga.

Para finalizar, añadiremos que el usuario del lenguaje FORTRAN tiene a su disposición un conjunto de rutinas con las que automáticamente puede realizar operaciones con los caracteres y las series de caracteres. Serán estudiadas en el Capítulo 7, 7.4. Funciones Intrínsecas.

6.7. SENTENCIA PARAMETER

La sentencia PARAMETER, que únicamente existe en el standard del FORTRAN completo, se utiliza para adjudicar un nombre simbólico a una constante.

Si recordamos la sentencia de entrada READ, uno de sus objetivos era el leer datos del exterior, constantes, y almacenarlos en determinadas posiciones de memoria con un nombre simbólico. Desde este punto de vista las sentencias READ y PARAMETER son equivalentes.

No obstante la diferencia está en que toda constante ingresada mediante una sentencia READ puede alterar su valor o contenido en el mismo programa, mientras que el valor ajudicado a una constante, al emplear una sentencia PARAMETER, ya no puede alterarse posteriormente.

Por ejemplo:

```
PARAMETER (PI = 3.141592)
```

adjudica el nombre simbólico PI a la constante 3.141592 y este valor para PI no puede variarse a lo largo de la ejecución del programa.

Estos elementos de datos definidos de esta forma, se denominan *nombres constantes*, mientras que los datos cuyo valor puede variar les hemos llamado hasta aquí, *nombres variables* o simplemente *variables*.

En una misma sentencia PARAMETER pueden definirse varios nombres constante. Por ejemplo, sería válido escribir:

```
PARAMETER (PI = 3.1415, CONT = 18.23,  E = 2.927)
```

Cuando un cierto valor numérico aparece varias veces en un programa es aconsejable adjudicarle un nombre mediante el uso de la sentencia PARAMETER y utilizar luego el nombre cada vez que sea necesario. Esta es una de las ventajas del uso de esta sentencia.

Un nombre constante puede emplearse en cualquier lugar de un programa FORTRAN, excepto en las especificaciones de formato.

La forma general de una sentencia PARAMETER es

```
PARAMETER (nombre1 = valor1, nombre2 = valor2, ...)
```

siendo: $nombre_i$ un nombre simbólico del valor $valor_i$

$valor_i$ es una expresión constante.

Si $nombre_i$ es de tipo entero, real, doble precisión o complejo, la expresión correspondiente, $valor_i$, debe ser una expresión constante aritmética.

Si $nombre_i$ es de tipo lógico o *carácter*, la expresión $valor_i$ que corresponde debe ser expresión constante lógica o *carácter*, respectivamente.

Otro ejemplo sería:

```
CHARACTER*10 A,B
PARAMETER (PI = 3.14, A = 'VALOR_DE_9' , B = ' (_DECIMO_) ')
```

Cualquier nombre simbólico que aparezca en *valor*$_i$ tiene que ser previamente definido con un valor en la misma sentencia PARAMETER o en otra anterior. Por ejemplo no sería válido escribir

```
PARAMETER (A = 3 * H)
```

a menos que se hiciera

```
PARAMETER (H = 2.8520, A = 3 * H)
```

O bien, aunque no es aconsejable

```
PARAMETER (H = 2.8520)
PARAMETER (A = 3 * H)
```

He aquí dos reglas a observar en el manejo de esta sentencia:

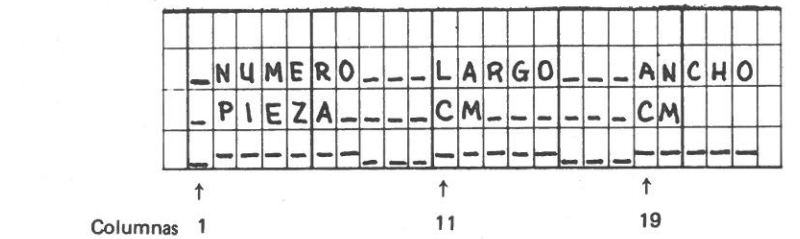
- Cualquier *nombre*$_i$ empleado en una sentencia PARAMETER tiene que ser declarado explícitamente mediante una sentencia de tipo previa a la sentencia PARAMETER donde se define su valor.
- La asignación de un *valor*$_i$ a una constante en una sentencia PARAMETER, debe preceder a cualquier uso que se haga en el programa del nombre constante.

Según estas reglas, el ejemplo último debería escribirse así:

```
REAL A,H
PARAMETER (H = 2.8520, A = 3 * H)
```

Programa 10. Se trata de crear una pequeña rutina para imprimir cabeceros o títulos. Para ello escribiremos el programa CABEC1 y el CABEC2. El segundo de ellos hace aplicación de la sentencia PARAMETER, y ambos, de algunos códigos de formato como el T y el A y de las constantes *carácter.*

Supongamos que el cabecero o titular a imprimir fuera:



es decir tres líneas consecutivas: la primera con las constantes *carácter* NUMERO, LARGO y ANCHO separadas cada una de la siguiente por tres blancos y dejando otro delante de NUMERO; la segunda línea contiene las constantes PIEZA, CM y CM precedidas también por un espacio en blanco y alineadas verticalmente con las anteriores; la tercera línea contiene el signo especial guión (–) para producir un subrayado y según el número total de caracteres que poseen los títulos NUMERO, LARGO y ANCHO.

El primer programa es el siguiente:

```
      PROGRAM CABEC1
      PRINT '(3(T2,A,T11,A,T19,A/))',
     1       'NUMERO','LARGO','ANCHO',
     2       'PIEZA','CM','CM',
     3       '------','-----','-----'
      STOP
      END
```

Figura 6.1

La sentencia PRINT contiene una constante *carácter* donde se definen los códigos de formato correspondientes, empleando para ello, a su vez, otra constante *carácter*. Obsérvese que se hace uso del código T para producir una tabulación. Todos los formatos se repiten tres veces y para imprimir tres líneas consecutivas se ha empleado el código / (slash).

El segundo programa CABEC2 es similar al anterior, pero en esta ocasión se ha hecho uso de la sentencia PARAMETER para definir la constante SERIE que contiene todos los formatos. Esta constante tiene una longitud de 22 caracteres.

```
      PROGRAM CABEC2
      CHARACTER*22 SERIE
      PARAMETER (SERIE='(3(T2,A,T11,A,T19,A/))')
      PRINT SERIE,'NUMERO','LARGO','ANCHO',
     1            'PIEZA','CM','CM',
     2            '------','-----','-----'
      STOP
      END
```

Figura 6.2

La sentencia PRINT actúa en este caso según el formato SERIE descrito en PARAMETER. Este formato podría variarse sin más que cambiar el contenido de dicha sentencia.

6.8. SENTENCIA DATA

El lenguaje FORTRAN 77 ha heredado de versiones anteriores, como FORTRAN 66, la posibilidad de utilizar en un programa la sentencia DATA, con la cual es posible adjudicar o asignar valores iniciales a variables, conjuntos, a elementos de un conjunto o a series de caracteres. Esta sentencia es una sentencia no ejecutable, pertenece al FORTRAN básico y al completo y puede aparecer en cualquier lugar de un programa pero en cualquier caso después de las sentencias de especificación.

De acuerdo con esta introducción, las sentencias DATA y PARAMETER parecen semejantes. Pero la diferencia entre ellas está en que, como ya sabemos, el valor adjudicado mediante el uso de la sentencia PARAMETER no puede ya ser modificado a lo largo del programa; los valores adjudicados a las variables mediante una DATA pueden variarse a lo largo de la ejecución del programa.

Los valores inicialmente definidos mediante DATA son realmente adjudicados a las variables correspondientes cuando comienza la ejecución del programa FORTRAN.

Supongamos que tuviéramos las siguientes sentencias de asignación aritmética:

```
A=5.6
B=7.0
X(4)=8.2
I=4
```

Entonces, haciendo uso de esta nueva sentencia, podríamos obtener el mismo efecto escribiendo, por ejemplo:

```
DATA   A, B, X(4), I / 5.6, 7.0, 8.2, 4/
```

sentencia que, por otra parte, es equivalente a esta otra:

```
DATA   A, B, X(4) / 5.6, 7.0, 8.2 /, I / 4 /
```

en la cual se han agrupado por una parte las variables reales y por otra la entera, con sus valores respectivos. Cuando el programa se cargue en la máquina para ser ejecutado, las variables tomarán los valores correspondientes contenidos entre los *slashes*. Obsérvese la coma (,) que separa ambos conjuntos de variables reales y enteras.

Cuando un mismo valor se adjudica a varias variables, entonces la sentencia DATA permite el uso de un coeficiente, constante entera sin signo y no nula, como indicativo de repetición. Por ejemplo:

```
DATA   A, B, C, D / 3*5.0, 7.2/
```

significa que las variables A, B y C se van a inicializar con el valor 5.0; el valor de D con 7.2.

Las sentencias:

```
DIMENSION SUM(20)
DATA SUM / 20*0.0 /, I, J, K, L / 3*5, 2 /
```

sirven para inicializar con ceros un conjunto unidimensional SUM de veinte elementos; para inicializar con el valor 5 tres variables más: las I, J y K y con el valor 2, la variable L.

Otro ejemplo sería:

```
LOGICAL A, B
DATA I, J, K / 3, 5, 7 /, A, B / 2 * . TRUE . /
```

con el cual las variables lógicas A y B tomarían el valor . TRUE. .

De la misma forma que hicimos en 3.12 al emplear el DO en las listas de variables de E/S, puede hacerse ahora dentro de una sentencia DATA. Por ejemplo, si fuera necesario realizar las asignaciones

```
J(1) = 10
J(2) = 20
J(3) = 30
```

ello podría lograrse escribiendo

```
DATA (J(I), I = 1,3) / 10, 20, 30 /
```

lo cual corresponde al uso del DO implícito dentro de una sentencia DATA.

Para inicializar a ceros un conjunto JOTA de tres dimensiones con un total de 125 elementos (= 5 x 5 x 5), podría escribirse la siguiente sentencia DATA que hace uso de un DO implícito:

```
DATA (((JOTA (I, K, L), I = 1,5), K = 1,5), L = 1,5) / 125 * 0 /
```

El uso del DO implícito no está permitido en el FORTRAN básico.

Debe observarse que todos aquellos datos que sean enteros deben inicializarse con valores enteros; los reales con reales; complejos con complejos y los definidos como lógicos o *carácter* con valores del mismo tipo equivalente.

Otro ejemplo sería:

```
INTEGER CONTA
CHARACTER NOM * 5
DATA A, CONTA, B, NOM / 0.0, 3, –7.2, 'MAPAS' /
```

Si el número de caracteres con el que se inicializa una variable *carácter* fuera menor que el declarado en la sentencia CHARACTER, entonces la variable se rellena con blancos a la derecha; si el número de caracteres de inicialización fuera mayor que la longitud declarada, entonces se pierden los caracteres de más a la derecha la variable. Asi por ejemplo

```
CHARACTER NOM *7, MINIM * 3
DATA NOM, MINIM / 'MAPAS', 'REDONDO' /
```

daría lugar a la siguiente asignación *carácter:*

```
NOM = 'MAPAS__'
MINIM = 'RED'
```

Por último, el siguiente ejemplo también sería válido:

```
CHARACTER*5 A, B(4)
DATA A / 'METRO' /, B(1) (1:2), B(1) (3:5) / 'CA', 'SAR' /
```

que asignaría 'METRO' a la variable A y CASAR a la variable B(1), primer elemento del conjunto de cuatro denominado B.

Con lo mostrado hasta aquí, la forma general de una sentencia DATA sería:

```
DATA variables1 / valores1 / , variables2 / valores2 / , ....
```

siendo: *variables*$_i$ una lista con nombres de variables, nombres de conjuntos, nombres de elementos de un conjunto, nombres de series de caracteres e incluso DO implícitos.

valores$_i$ una lista con valores constante, lógicos o *carácter* precedidos o no de un coeficiente de repetición.

Una última observación a tener en cuentra al utilizar la sentencia DATA es que el número total de nombres de variables de *variables*$_i$ debe coincidir con el de valores contenidos en *valores*$_i$.

6.9. SENTENCIA GO TO ASIGNADO

Como ya hemos visto, el lenguaje FORTRAN dispone de distintas instrucciones ejecutables para realizar bifurcaciones o saltos en un programa: IF, GO TO, etc.

Algunas de ellas son muy útiles como el GO TO calculado (2.10) y otras menos como el IF aritmético (2.11) o el GO TO asignado que ahora vamos a estudiar.

Esta sentencia está siempre relacionada con la ASSIGN que debe aparecer en el mismo programa que el GO TO asignado.

La forma general o sintáxis de esta dos sentencias es

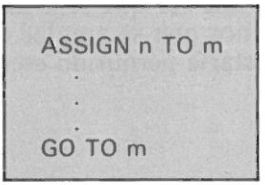
```
ASSIGN n TO m
      .
      .
      .
GO TO m, (n1, n2, ....., nn)
```

siendo: n el número o etiqueta de una sentencia existente en el programa.

m es el nombre de una variable entera.

n_i son etiquetas o números de sentencias ejecutables existentes en el mismo programa FORTRAN. Dentro del paréntesis puede repetirse más de una vez cualquiera de los valores n_i.

Obsérvese que puede faltar la coma (,) y el paréntesis con las n_i después del valor m de la sentencia GO TO; de modo que otra forma más simplificada sería:



Por ejemplo:

```
ASSIGN  100 TO I
   .
   .
   .
   .
GO TO I
```

asigna el número 100, que corresponde a una etiqueta dentro del programa, a la variable I. Más adelante tendrá lugar la bifurcación

```
GO TO 100
```

Si la codificación hubiera sido

```
ASSIGN  100  TO  I
   .
   .
   .
   .
GO TO I, (103, 104, 100)
```

el efecto habría sido equivalente al del ejemplo anterior; pero en este caso la sentencia GO TO contiene otros números de sentencias ejecutables, contenidas en el mismo programa y a las cuales sería posible bifurcar dependiendo del valor que tomara I mediante ASSIGN.

Ténganse en cuenta las siguientes observaciones:

- El valor de m debe coincidir con alguno de los valores n_i contenidos en los paréntesis del GO TO, al utilizar la primera forma general.
- El valor de m no puede ser alterado una vez que se le ha asignado un valor mediante la ASSIGN a menos que se emplee una nueva sentencia ASSIGN. Según esto, no estaría permitido escribir:

```
ASSIGN 200 TO N
N = N + 2
GO TO N, (100, 200, 202)
```

Pero si podría haberse escrito:

```
ASSIGN 200 TO N
   . . .
ASSIGN 202 TO N
   . . .
GO TO N, (100, 200, 202)
```

La sentencia ASSIGN puede también utilizarse para asignar una constante entera a una variable que luego figure como etiqueta en una sentencia de E/S. Por ejemplo:

```
ASSIGN 100 TO IFOR
READ (1, IFOR) A, B, C
```

equivale a leer por la unidad 1 según el formato 100 (etiqueta de un FORMAT).

EJERCICIOS RESUELTOS

6.1. Empleando una única sentencia de tipo, convertir en reales las siguientes variables y conjuntos: I, JOTA(5), KM, LOR(8, 3, 10)

Respuesta

```
REAL I, JOTA(5), KM, LOR(8, 3, 10)
```

Otra alternativa menos simple, sería:

```
DIMENSION JOTA(5), LOR (8, 3, 10)
REAL I, JOTA, KM, LOR
```

6.2. Convertir en enteros los siguientes conjuntos y variables: ALFA(2, 3), X, Y, OMEGA(80). Emplear una declaración de tipo.

Respuesta

```
INTEGER ALFA(2, 3), X, Y, OMEGA(80)
```

Otra solución:

```
DIMENSION ALFA(2, 3), OMEGA(80)
INTEGER ALFA, X, Y, OMEGA
```

6.3. Indicar el significado algebráico de las siguientes constantes complejas FORTRAN:

a) (8.2E+1, –2.0)
b) (–45, +3)
c) (202., –2.9E1)
d) (0., 0.)
e) (4,5)
f) (0.0E+0, 0.1E+00)

Respuesta

a) 82 –2i
b) –45 + 3i
c) 202 –29i
d) 0 + 0i
e) 4 + 5i
f) 0 + 0,1i

6.4. Las siguientes variables o conjuntos están formados por elementos complejos.

Definirlos como tales para el compilador: I, ALFA(3,8), JOTA, OMEGA(3).

Respuesta

```
COMPLEX I, ALFA(3,8), JOTA, OMEGA(3)
```

O bien:

```
DIMENSION ALFA(3,8), OMEGA(3)
COMPLEX I, ALFA, JOTA, OMEGA
```

6.5. Dada la codificación

```
COMPLEX A(3,3), B
DOUBLE PRECISION C, D(2,2)
IMPLICIT REAL (E – G)
IMPLICIT INTEGER (W – Z)
```

deducir los tipos de las siguientes expresiones aritméticas:

a) (A(2,3) * EL) + GAMA
b) WATIO * (C + 2 * B)
c) (ERNA + ZETA) / D(1,2) *¡ A(3,2)
d) (FI * XILO) + ((B + A(1,1)) * 2) / 3 . 5
e) (W + Z) * C
f) (4.5 – WA) / E2
g) (WATIO – YEL) / 2

Respuesta

a) Compleja
b) No está permitido mezclar tipos real y complejo
c) No está permitido mezclar tipos real y complejo
d) Compleja
e) Doble precisión
f) Real
g) Entera

6.6. Dadas las declaraciones siguientes.

```
COMPLEX A, B
DOUBLE PRECISION C, D, E
REAL F, G
```

encontrar el tipo resultante para las expresiones aritméticas siguientes

a) (F ** C) + A
b) (F – G) / 4.8 + (E ** F)
c) (A ** B) + F / 3
d) 1.0 + A ** 2.0
e) E / 2 * F

Respuesta

a) No está permitida la mezcla doble precisión-complejo.
b) Doble precisión.
c) Complejo.
d) Complejo.
e) Doble precisión.

6.7. Dada la codificación

```
DOUBLE PRECISION A
COMPLEX B, C, D, E
I=(3.2, 3.7)
JOTA = (2.2, –4,5)
Z=(4.8, 5.3)
A=(5., –6.2)
B=5
C=–2.3
D=3.9+;E00
E=(–5.5, –6.7)
```

deducir los contenidos de I, JOTA, Z, A, B, C, D y E una vez ejecutadas las sentencias de asignación aritmética que se encuentran en la codificación anterior.

Respuesta

I = 3 " JOTA = 2 " Z = 4.8 " A = 5 (doble precisión) " B = (5.0, 0.0)
C = (–2.3, 0.0) " D = (3.5, 0.0) " E = (–5.5, –6.7)

6.8. Dadas las sentencias

```
      REAL I, J
      COMPLEX A, B, C, D
      READ (1, 100), A, I, C, D, J
100   FORMAT (2X, F10.2, F5.1, E14.8, F5.2, E14.8, E15.8, E14.7, 3X, F10.3)
```

deducir los códigos de formato asociados con cada una de las variables de la lista de entrada.

Respuesta

A con F10.2 y F5.1
I con E14.8
C con F5.2 y E14.8
D con E15.8 y E14.7
J con F10.3

6.9. Dada la codificación.

```
      COMPLEX A
      READ (1,100) A
100   FORMAT (F5.2, F7.3)
```

y suponiendo que la información introducida por el terminal pantalla-teclado fuera

```
-3502433.82074
```

¿Cuál sería el número complejo almacenado en A?

Respuesta

$A = (-35.02, 433.820) \equiv -35{,}02 + 433{,}82\ i$

6.10. Supuesto que es $A = 5{,}0 - 7{,}2i$ y $B = 2{,}1 + 7{,}0i$ escribir en FORTRAN las expresiones complejas:

a) $A + B$
b) $A^2 + B^2$
c) $A + B^2$
d) $B - A^2$

Respuesta

```
a) (5.0, -7.2) + (2.1, 7.0)
b) (5.0, -7.2) ** 2 + (2.1, 7.0) ** 2
c) (5.0, -7.2) + (2.1, 7.0) ** 2
d) (2.1, 7.0) - (5.0, -7.2) ** 2
```

6.11. Se posee un conjunto de variables *carácter* unidimensional de 50 elementos CONJ. Cada una de las variables tiene una longitud de 10 caracteres. Escribir las declaraciones correspondientes para el compilador.

Respuesta

```
CHARACTER*10  CONJ(50)
```

o bien

```
CHARACTER*10, CONJ
DIMENSION  CONJ(50)
```

6.12. Dada la relación

CONJUNTO	DIMENSIONES	LONGITUD
JOTA	5 x 3	15 caracteres
BETA	2 x 5	6 "
ZETA	5 x 1	6 "
OMEGA	6 x 1	7 "
ROSA	4 x 3	15 "

escribir las correspondientes declaraciones FORTRAN

Respuesta

```
CHARACTER*15  JOTA(5,3), ROSA(4,3), BETA(2,5)*6,
              ZETA(5)*6, OMEGA(6)*7
```

o bien

```
CHARACTER*15 JOTA, ROSA, BETA*6, ZETA*6, OMEGA*7
DIMENSION JOTA(5,3), ROSA(4,3), BETA(2,5), ZETA(5), OMEGA(6)
```

6.13. Supuesto que SERIE vale

```
SERIE = 'ESTUDIO_MODULAR_NUMERO_4'
```

deducir los contenidos de las series de caracteres SER y SERIE que se definen, indicando si alguna de las asignaciones no es válida.

a) SER = SERIE (1:6) // SERIE (10:)
b) SER = SERIE (8:12)
c) SERIE = SERIE (2:4) // SERIE (15:18)
d) SER = SERIE (:10) // SERIE (2:3)
e) SERIE = SERIE (:20)
f) SER = SERIE (3:4) // SERIE (7:10)
g) SERIE = 'ALFA' // SERIE (2:5)

Respuesta

a) 'ESTUDIODULAR_NUMERO_4'
b) '_MODU'
c) No permitida

d) 'ESTUDIO_MOST'

e) No permitida

f) 'TUO_MO'

g) No permitida

6.14. Dada la serie anterior SERIE, deducir una codificación para poder asignar a SERIE la serie concatenada SERIE(1:5) // SERIE (7:10) perteneciente a ella misma y que no está permitido hacer directamente ¿Cuál será el contenido último de SERIE?

Respuesta

```
CHARACTER*24 SERIE, INTER*9
SERIE = 'ESTUDIO_MODULAR_NUMERO_4'
INTER = SERIE (1:5) // SERIE (7:10)
SERIE = INTER
```

El contenido último de SERIE será:

```
SERIE = 'ESTUDO_MO_____________'
```

6.15. Dada la serie de caracteres STRING, cuyo contenido es

```
STRING = 'LOCALIZACION_DE_LOCOMOTORAS_LOCALES'
```

de una longitud N = 35 caracteres, escribir la codificación conveniente para contabilizar el número total de veces que aparece en STRING la serie 'LO'

Respuesta

```
      N = 35
      I = 0
      DO 1 M = 1, N-1
      IF (STRING (M:M+1) . EQ . 'LO') I = I+1
    1 CONTINUE
```

El total de veces que aparece 'LO' estará contenido en la variable I.

6.16 Dada la misma serie anterior, escribir otra codificación para contabilizar el total de caracteres no blancos que posee STRING. Almacenar ese total en una variable LONG.

Respuesta

```
      DO 1 LONG = 35,0, -1
      IF (STRING (LONG: LONG) . NE . '_') GO TO 2
    1 CONTINUE
      LONG = LONG + 1
    2 CONTINUE
```

6.17. Dada la codificación

```
DIMENSION  LOG(3)
LOGICAL  LOG
DOUBLE  PRECISION
COMPLEX  B
DATA (LOG(I), I=1,3) / 2 * . TRUE ., . FALSE . /, A, B / 7.22, (2.3, -1.)/
```

Decir qué valores serían adjudicados a las variables:

a) LOG(3), LOG(1)
b) A
c) B

Respuesta

a) LOG(3) = . FALSE .
   LOG(1) = . TRUE .

b) A = 7.22 en doble precisión

c) B tomará el valor complejo 2,3 −1,0i

6.18. Dada la codificación

```
CHARACTER*8  K, Y
DATA  K, Y / '2468' , 'ABCDEFGHIJK'/
```

¿Cuáles serán los valores adjudicados a K e Y?

Respuesta

K = '2468____'
Y = 'ABCDEFGH'

6.19. Decir los errores, si existen, contenidos en las sentencias siguientes:

a) DATA I, J, A / 3. E+02, 3, 4.7 /
b) DATA A, B, C / 3*8.2, –4.5 /
c) DATA A, I, J, C / –5.2, 8, 3 /

Respuesta

a) A la variable I se le adjudica un valor real
b) La lista de variables contiene tres datos y la de constantes cuatro
c) La lista de variables contiene cuatro datos y la de valores solo tres

6.20. Dada la codificación

```
LOGICAL  A(5.2)
DATA  A/3*. TRUE., 5*. FALSE., 2*. TRUE./
```

qué valores se asignan a las variables A(1,1), A(2,1), A(4,2), A(5,2) y A(3,1).

Respuesta

Como los conjuntos se almacenan por columnas, es decir, variando más rápidamente el primer subíndice, será:

A(1,1) = . TRUE .
A(2,1) = . TRUE .
A(4,2) = . TRUE .
A(5,2) = . TRUE .
A(3,1) = . TRUE .

6.21. Dada la codificación

```
CHARACTER*4  A(3)
DATA  A/ 'ROJO', 'MAR', 'CINCO' /
```

¿cuáles serán los valores adjudicados a cada variable del conjunto A?

Respuesta

A(1) = 'ROJO'
A(2) = 'MAR_'
A(3) = 'CINC'

6.22. Encontrar los errores, si existen, de las siguientes sentencias:

a) GO TO M, (100, 200, 300)
b) GO TO 20, (20, 15, 10)
c) GO TO I, (M, N, J)
d) ASSIGN M TO N
e) ASSIGN 300 TO JOTA
f) ASSIGN –74 TO N

Respuesta

a) Es correcto
b) Después de GO TO debe existir una variable entera
c) Los paréntesis deben contener números de sentencia
d) M debe ser el número de una sentencia, no una variable
e) Es correcto
f) El número de una sentencia no puede ser negativo

7

Programación Modular: Funciones

Es contrario a la razón decir
que existe un vacío o un espacio
donde no existe absolutamente nada

DESCARTES *(1596-1650)*

7.1. UNIDADES PROGRAMA

Al escribir un programa para un ordenador en un lenguaje cualquiera, sucede con frecuencia que una misma sentencia o grupo de ellas se precisan en distintos lugares del programa. Esta situación se presentaría, por ejemplo, si fuera necesario calcular la raíz cúbica en distintas ocasiones dentro de la misma codificación.

Igualmente y cuando los programa son muy complejos o contienen un gran número de sentencias, es muy común y conveniente subdividirlos en partes más pequeñas de manera que cada una de ellas resuelvan un algoritmo específico, para a continuación y con todas ellas procesar el programa total como una unidad.

Estos dos ejemplos, grupos de sentencias que se repiten o subdivisión de un programa en partes, nos conducen al concepto de módulo o *unidad programa* FORTRAN, entendiendo por tal un conjunto de sentencias y, opcionalmente, también líneas de comentario.

Todos los programas ejemplo estudiados hasta aquí dentro de este texto, son ejemplos de unidad programa o módulo y debido a que pueden ser procesados de una forma autónoma e independiente de cualquier otro, es por lo que también reciben el nombre de *programa principal.*

Existen otros módulos o unidades que únicamente pueden ser procesados cuando son llamados a ejecutarse por otros: estos constituyen la segunda categoría de módulos denominada *subprogramas* y son conjuntos de instrucciones de uso general asociados a un programa principal.

Resumiendo: un *módulo* puede ser un programa principal o un subprograma.

La utilización de distintos módulos, programas principales y subprogramas, a la hora de escribir un programa, constituye una técnica comúnmente conocida con el nombre de *programación modular*. Esta técnica contribuye a beneficiar la escritura de programas debido a la simplicidad y eficiencia que se logran con su empleo.

Con los conceptos introducidos hasta aquí, el diseño y desarrollo de un programa FORTRAN podría realizarse siguiendo las siguientes etapas principales:

- Definición del problema que va a resolverse y codificarse.
- Realización de un diseño completo de la estructura modular del programa, principalmente cuando éste es extenso o divisible.
- Definición de los datos, de los resultados a obtener y de la estructura lógica que debe poseer cada uno de los módulos o unidades programa.
- Estudiar la posibilidad de que un determinado módulo pueda a su vez ser subdividido o modularizado.
- Codificación del programa haciendo uso de sentencias FORTRAN.

Otro concepto que vamos a describir es el de la conexión entre subprogramas y programas principales. Los subprogramas pueden conectarse al principal de dos formas distintas:

- Incorporándose al programa principal objeto, cada vez que se haga referencia a él en el fuente. Se trata entonces de un *subprograma abierto* o *en línea.*
- Incorporándose una sola vez al programa principal objeto independientemente del número de veces que sea llamado por este. Se trataría de un *subprograma cerrado* o *fuera de línea.*

Ambas incorporaciones son realizadas automáticamente por el propio sistema y el programador o usuario no debe estar pendiente de esta mecánica.

7.2. SUBPROGRAMAS

Hemos dicho anteriormente que los ejemplos de programas explicados, constituían módulos especiales denominados programas principales. Para el compilador estos módulos están perfectamente identifi-

cados por comenzar por la sentencia PROGRAM o bien por otra cualquiera que no tenga un sentido o significación especial como SUBROUTINE, FUNCTION o BLOCK DATA que serán estudiadas en detalle más adelante.

Los programas principales comienzan su ejecución simplemente dando al sistema una orden adecuada como por ejemplo:

```
EJECUTAR nombre
```

siendo *nombre* la denominación de dicho programa, aunque esta orden no pertenece al lenguaje FORTRAN, sino que es un comando del propio sistema.

Hemos dicho también que los subprogramas, con frecuencia denominados *procedimientos*, son siempre llamados a ejecución por un módulo principal.

Con todos estos conceptos estamos en condiciones de definir ya lo que es un *sistema de programas*. Se entiende por tal, el conjunto formado por un programa principal y uno o varios subprogramas de manera que todos ellos constituyen una unidad para resolver un problema determinado.

Normalmente cada uno de los subprogramas se codifican para resolver con ellos un segmento lógico o algorítmico dentro del problema total. Es el programa principal el que gobierna la ejecución de cada módulo y cuándo debe comenzar ésta.

Los subprogramas pueden ser de dos clases: *subrutinas* y *funciones* y cada uno de ellos va a ser descrito a continuación. Una forma esquemática de representar el funcionamiento y unión entre programa principales y subprogramas queda reflejada en la figura 7.1

7.3. SUBRUTINAS Y FUNCIONES

La subrutina es una de las herramientas más útiles de las que dispone un usuario del FORTRAN. Con ellas se puede formar un sistema de programas y constituyen el medio más adecuado con el que lograr una modularización correcta.

Toda subrutina es un módulo separado del principal, por lo que precisa de una primera sentencia especial para poder ser reconocido por el compilador o por el sistema. Esta sentencia es

```
SUBROUTINE
```

y será estudiada con todo detalle más adelante.

Una subrutina únicamente se ejecuta cuando se llama desde el programa principal utilizando la sentencia

CALL

escrita en dicho programa. Esta sentencia además de establecer la unión entre módulo principal y subrutina, le transmite a ésta los datos denominados *argumentos* con los que debe ejecutar los cálculos que en ella se encuentran codificados. Mientras tiene lugar la ejecución de la subrutina queda suspendida la ejecución del módulo principal volviendo a iniciarse ésta cuando se *retorna* el control desde la subrutina a aquel módulo y precisamente en el punto o sentencia siguiente de donde se partió. Véase de nuevo la figura 7.1.

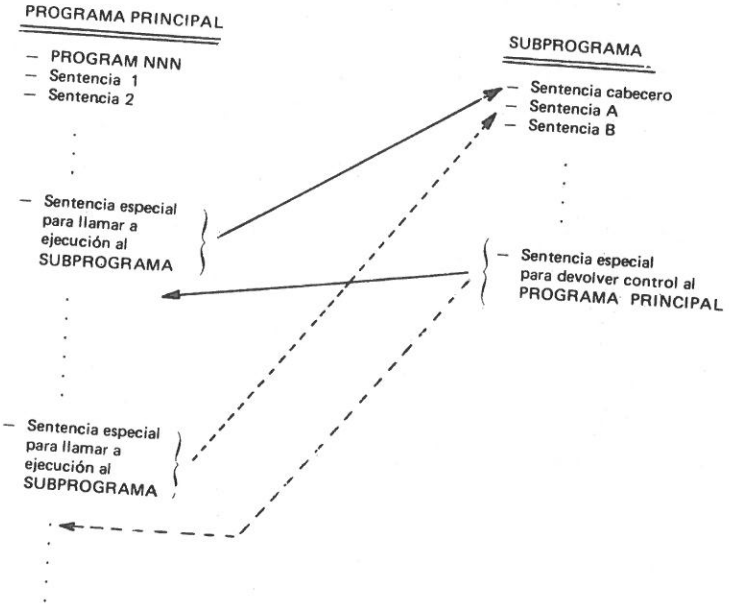


Figura 7.1

Cuando la subrutina devuelve control al programa principal, le entrega a éste, en general, varios valores, resultado de los cálculos que se han realizado durante su proceso.

La otra forma de producir una modulación es mediante el uso de funciones FORTRAN. Las funciones son semejantes a las subrutinas en cuanto que constituyen módulos separados de un programa principal. La diferencia entre unas y otras es que las funciones devuelven un solo valor al programa principal y este valor es del mismo tipo, entero, real, etc., que el tipo al que pertenece el propio nombre de la función.

Para llamar a una función y que entre en proceso, es necesario dar su nombre desde el programa principal y, como en el caso de las subrutinas, la lista de datos o *argumentos* con los que va a trabajar. El nombre de la función debe formar parte, como operando, de una expresión contenida, a su vez, en el programa principal, cuando se la llame para ser procesada.

Una vez finalizada la ejecución de una función, el control se devuelve al programa principal y concretamente al operando de la expresión desde donde se llamó, de manera que el valor único calculado por la función sustituye a dicho operando continuándose la evaluación del resto de la expresión.

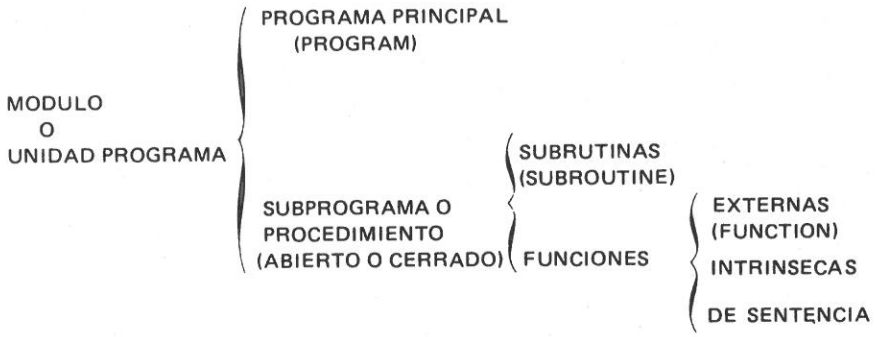
En FORTRAN existen tres clases de *funciones: externas, intrínsecas* y *de sentencia*. Las dos primeras son programas separados del principal: las externas las programa o realiza el propio usuario mediante el empleo de sentencias FORTRAN, mientras que las segundas están preescritas y son facilitadas por el propio sistema FORTRAN. De estas últimas ya hemos visto algunos ejemplos (véase *Programa 9)* anteriormente. Así, para calcular la raíz cuadrada de un número se emplea una función intrínseca cuyo nombre es SQRT; el valor absoluto de una catidad se logra con ABS; el logaritmo natural con LOG; el máximo de un conjunto de valores con MAX, etc.

Las funciones externas se llaman o ejecución escribiendo el nombre que el programador les haya dado, acompañado de la palabra

FUNCTION

La tercera categoría de funciones, es decir, las *funciones de sentencia,* se denominan así por estar constituidas únicamente por una sola sentencia FORTRAN. Estan siempre incorporadas o en línea dentro de un módulo por lo que son locales o privativas de ese módulo.

El resumen de todo lo dicho hasta aquí, queda reflejado en el siguiente esquema:



7.4. FUNCIONES INTRINSECAS

Ya nos hemos referido anteriormente a este tipo de módulos. Vamos ahora a hablar de ellos con más detalle iniciando así un tema amplio y útil como es el de los subprogramas o procedimientos.

Todas las funciones intrínsecas constituyen un paquete o conjunto que el sistema FORTRAN pone a disposición de los usuarios y la mayor parte de ellas son ampliamente utilizadas, como por ejemplo, las funciones trigonométricas, exponenciales, logarítmicas o las relacionadas con el manejo de caracteres.

Cada una de las funciones intrínsecas posee un *nombre*, un *tipo* y contiene uno o más *argumentos* que son los valores con los que la función trabaja y realiza los cálculos.

Supongamos el ejemplo:

```
SQRT (5.7)
```

Cuando esta construcción aparece en un módulo cualquiera FORTRAN, contenida dentro de una expresión aritmética como un operando, da lugar a que se *llame* a la función precisamente programada encargada de calcular la raíz cuadrada de la expresión encerrada entre paréntesis.

En este caso, el nombre de la función es SQRT; su tipo es real, porque una vez realizado el cálculo de encontrar la raíz cuadrada de 3.0,

devuelve como resultado un valor real; en esta ocasión, la función únicamente trabaja con un argumento: el valor 3.0, cuya raíz cuadrada ha de calcularse.

El resultado de una función intrínseca sustituye al operando desde donde fue llamada. Por ejemplo, si tuvieramos

```
A = B + SQRT (7.4)
```

entonces, el operando SQRT (7.4) de la expresión B + SQRT (7.4) originaría el cálculo de la raíz cuadrada del argumento real 7.4 y una vez encontrado el resuldado, que también será de tipo real, ese valor será añadido al contenido de la variable B. El resultado total será asignado a A.

La forma general de estas funciones es

```
NOMBRE (a1, a2, ... an)
```

siendo NOMBRE una denominación preestablecida por el sistema y a_1, a_2, ..., a_n, los argumentos sobre los que trabajará la función intrínseca llamada.

Un argumento puede estar constituido por una constante, una variable, por un elemento de un conjunto o por una expresión, pero siempre que cada uno de esos elementos responda a la exigencia sobre su tipo que imponga la función.

En los cuadros que siguen se indica con detalle esta información.

Otra característica a tener en cuentra es la de que los argumentos no cambian su tipo o valor cuando se emplean en la llamada a una función. Por ejemplo

```
Y = ABS (JOTA)
```

calculará el valor absoluto del contenido de JOTA, que luego será asignado a la variable Y. Pero realmente el contenido previo de JOTA no se verá alterado por ese cálculo.

Por último diremos que estas funciones pueden emplearse en nido o cascada, es decir, se puede escribir como argumento de una función a su vez otra función. Así por ejemplo

```
A = SQRT (SIN (X))
```

calcuaría, en primer lugar, el seno trigonométrico del valor contenido en el argumento X y luego tomaría ese resultado como nuevo argumento para calcular su raíz cuadrada, valor que se asignaría a A.

Usualmente el valor calculado mediante una función intrínseca tiene el mismo tipo que sus propios argumentos, (que también son del mismo tipo entre sí cuando existen varios) salvo en algunas excepciones o funciones, las cuales se denominan *funciones de transferencia*, porque partiendo, por ejemplo, de un argumento real, se obtiene como resultado un valor entero. Estas funciones son: INT, NINT, REAL, CHAR, ICHAR y LEN.

En los cuadros que siguen se habla de *nombres genéricos* y de *nombres específicos* para denominar las funciones intrínsecas. Un *nombre genérico* es una única denominación que comprende varias clases de funciones intrínsecas. Por ejemplo MAX, es el nombre genérico con el que se denominan todas las funciones, MAX0, AMAX1, DMAX1, encargadas de calcular el valor más grande dentro de un conjunto de ellos. El *nombre específico* corresponde a cada función y está relacionado con el tipo de argumentos que se emplean en ella.

El programador puede emplear nombres genéricos o nombres específicos indistintamente. Esto quiere decir que cuando se emplea un nombre genérico (si existe) para llamar a una función intrínseca, el propio procesador sustituirá este nombre por el específico correspondiente dependiendo del tipo al que pertenezcan los argumentos de la función.

Por ejemplo, si el programador escribe

```
MAX (3.5, 2.7, 3.0, –1.4)
```

empleando MAX como nombre genérico, el procesador entenderá que se trata de una llamada a la función AMAX1 encargada del cálculo del máximo de un conjunto de valores reales. Por ello, se hubiera conseguido el mismo efecto, escribiendo

```
AMAX1 (3.5, 2.7, 3.0, –1.4)
```

Tipo de argumentos, indica el tipo FORTRAN de los mismos. El número que figura delante de ellos, significa la cantidad a emplear en la llamada, cuando esta es distinta de uno. *Tipo de función* indica el tipo del resultado obtenido una vez ejecutada. Las letras I, R, DP, C, CH y L indican, respectivamente, entero, real, doble precisión, complejo, *carácter* y lógico.

En los cuadros siguientes figuran todas las funciones standard del FORTRAN 77 completo. El lector debe consultar los manuales de su sistema cuando utilice un FORTRAN básico, en el cual, y como regla general, no existirán las funciones que traten con datos doble precisión o complejos.

NOMBRE GENERICO	NOMBRE ESPECIFICO	TIPO ARGTS	TIPO FUNC	DEFINICION
INT	–	I	I	Calcula la parte entera de un argumento, truncándolo si fuera necesario. Ejemplos: INT(5) = 5 , INT (3.5) = 3 INT(3.8, –2.3) = 3 (argumento complejo) INT equivale a IFIX
	INT	R	I	
	IFIX	R	I	
	IDINT	DP	I	
	–	C	I	
REAL	REAL	I	R	Calcula el valor real equivalente al valor del argumento. Ejemplos: REAL(8.2) = 8.2 , REAL(2.1, 5.7) = 2.1 REAL(3) = 3.0 FLOAT equivale a REAL
	FLOAT	I	R	
	–	R	R	
	SNGL	DP	R	
	–	C	R	
DBLE	–	I	DP	Calcula el valor doble precisión equivalente al valor del argumento. Ejemplos: DBLE(3.0D0) = 3.0D0 " DBLE(5) = 5.0D0 DBLE(7.2, 3.5) = 7.2D0
	–	R	DP	
	–	DP	DP	
	–	C	DP	
CMPLX	–	1 ó 2, I	C	Calcula el valor complejo equivalente al valor del argumento. Ejemplos: CMPLX(2.1, 0.7) = (2.1, 0.7) CMPLX(5) = (5.0, 0.0) CMPLX(0.7) = (0.7, 0.0)
	–	1 ó 2, R	C	
	–	1 ó 2, DP	C	
	–	1 ó 2, C	C	
	ICHAR	CH	I	El argumento está constituido por un carácter único y la función determina la posición de ese carácter dentro del conjunto de caracteres alfanuméricos ordenados según una determinada secuencia que cada máquina posee. Ejemplo: ICHAR (*) = 5 significa que el asterisco ocupa la posición 6 dentro de la sucesión de caracteres de una máquina conforme al sistema de codificación ASCII. La sucesión comienza en cero.

NOMBRE GENERICO	NOMBRE ESPECIFICO	TIPO ARGTS	TIPO FUNC	DEFINICION
	CHAR	I	CH	Obtiene como resultado el carácter que ocupa el lugar indicado por el argumento dentro de la secuencia ordenada de caracteres que posee cada máquina y que comienza en la posición cero. Ejemplos: CHAR(3) = (CHAR(6) = + CHAR (22) = A
AINT	AINT DINT	R DP	R DP	Producen un truncamiento del argumento x seguido de una conversación real o doble precisión del resultado obtenido, es decir, estas funciones equivalen a REAL (INT(X)) y DBLE (INT(X)) Ejemplos: AINT(5.7) = 5.0 DINT(8.3) = 8.0D0
ANINT	ANINT DNINT	R DP	R DP	Obtención del entero más próximo (expresado en forma real) al valor del argumento (redondeo). Si es $x \geqslant 0$, se calcula REAL(INT(X + 0.5)) ó DBLE (INT(X + 0.5)) Si es $x < 0$, se calcula REAL (INT(X−0.5)) ó DBLE (INT(X−0.5)) Ejemplos: ANINT(4.3) = 4.0 ANINT(−2.4D0) = −2.0DO ANINT(5.6) = 6.0
NINT	NINT IDNINT	R DP	I I	Obtención del valor entero más próximo al valor del argumento (redondeo). Si es $x \geqslant 0$, se calcula INT (X + 0.5) Si es $x < 0$, se calcula INT (X −0.5) Ejemplos: NINT (10.3) = 10 NINT (4.5 D0) = 5

NOMBRE GENERICO	NOMBRE ESPECIFICO	TIPO ARGTS	TIPO FUNC	DEFINICION
ABS	IABS ABS DABS CABS	I R DP C	I R DP R	Calcula el valor absoluto o módulo del argumento, es decir, si x_r y x_i son las partes reales e imaginarias del argumentos, se calcula $\sqrt{x_r^2 + x_i^2}$ Ejemplos: ABS (−5) = 5 " DABS(2.0D0) = 2.0D0 CABS(4.0, 3.0) = 5.0
MOD	MOD AMOD DMOD	2,I 2,R 2,DP	I R DP	Calcula el resto de dividir el primer argumento x_1 por el segundo x_2, es decir, siendo $x_2 \neq 0$, MOD(X1, X2) = X1 − (INT (X1/X2) * X2) AMOD(X1, X2) = REAL (X1 − (INT (X1/X2) * X2)) DMOD (X1, X2) = DBLE (X1 − (INT (X1/X2) * X2))
SIGN	ISIGN SIGN DSING	2,I 2,R 2,DP	I R DP	Transferencia de signo de un argumento x_1 al otro x_2, es decir: Si es $x_2 \geqslant 0$, se calcula ABS(X1) Si es $x_2 < 0$, se calcula −ABS(X1)
DIM	IDIM DIM DDIM	2,I 2,R 2,DP	I R DP	Calcula la diferencia positiva entre los dos argumentos x_1 y x_2, es decir X1−X2 si es $x_1 > x_2$ o cero en otros casos.
MAX	MAX0 AMAX1 DMAX1	$\geqslant$2,I $\geqslant$2,R $\geqslant$2,DP	I R DP	Calcula el máximo de un conjunto de valores o argumentos. Ejemplos: AMAX0 (3,−7, 4, −8) = 4.0 MAX (2.5, 6.2, −7.1) = 6.2
	AMAX0 MAX1	$\geqslant$2,I $\geqslant$2,R	R I	
MIN	MIN0 AMIN1 DMIN1	$\geqslant$2,I $\geqslant$2,R $\geqslant$2,DP	I R DP	Calcula el mínimo de un conjunto de valores o argumentos. Ejemplos: MIN1 (−1.5, 2.8, 03) = −1 MIN0 (4, 3, 2, −7, −9) = −9
	AMIN0 MIN1	$\geqslant$2,I $\geqslant$2,R	R I	

NOMBRE GENERICO	NOMBRE ESPECIFICO	TIPO ARGTS	TIPO FUNC	DEFINICION
	DPROD	2,R	DP	Dados dos argumentos reales, calcula su producto expresándolo en doble precisión
	LEN	CH	I	Obtiene la longitud de una constante *carácter*. Ejemplos: LEN('AMIGO') = 5 LEN ('AL_SOL') = 6
	INDEX	2,CH	I	INDEX (X1, X2). Obtiene la posición de la primera aparición de la serie de caracteres X_2 en la serie de caracteres X_1. Si la serie X_2 no aparece en la X_1, se devuelve el valor cero. Ejemplo: si fuera AX2 = 'AMANECER_CLARO' entonces, sería INDEX (AX2, 'CLA') = 10
	AIMAG	C	R	Obtiene la parte imaginaria de un argumento complejo. Ejemplo AIMAG(3.2, −7.5) = −7.5
	CONJG	C	C	Obtiene el complejo conjugado de otro. Ejemplos: CONJG(3.2, 2.7) = (3.2, −2.7) CONJG(1.5, 0.0) = (1.5, 0.0)
SQRT	SQRT DSQRT CSQRT	R DP C	R DP C	Calcula la raíz cuadrada del argumento que tiene que ser positivo
EXP	EXP DEXP CEXP	R DP C	R DP C	Calcula el valor de *e* elevado a la potencia indicada por el argumento, siendo *e* la base de los logaritmos naturales o neperianos.
LOG	ALOG DLOG CLOG	R DP C	R DP C	Calcula el logaritmo en base *e* o logaritmo natural del valor del argumento.
LOG10	ALOG10 DLOG10	R DP	R DP	Calcula el logaritmo en base 10 ó logaritmo común del valor del argumento.

NOMBRE GENERICO	NOMBRE ESPECIFICO	TIPO ARGTS	TIPO FUNC	DEFINICION
SIN	SIN DSIN CSIN	R DP C	R DP C	Calcula el seno del argumento que debe expresarse en radianes. Una circunferencia tiene 2π radianes.
COS	COS DCOS CCOS	R DP C	R DP C	Calcula el coseno del argumento que debe expresarse en radianes.
TAN	TAN DTAN	R DP	R DP	Calcula la tangente trigonométrica del argumento que debe expresarse en radianes.
ASIN	ASIN DASIN	R DP	R DP	Calcula la función arco seno del argumento, es decir, el ángulo en radianes que corresponde a un argumento seno.
ACOS	ACOS DACOS	R DP	R DP	Calcula la función arco coseno del argumento, es decir, el ángulo en radianes que corresponde a un argumento coseno.
ATAN	ATAN DATAN	R DP	R DP	Calcula la función arco tangente del argumento, es decir, el ángulo en radianes que corresponde a un argumento tangente
ATAN2	ATAN2 DATAN2	2,R 2,DP	R DP	Calcula la función arco tangente del cociente de los dos argumentos, es decir, el ángulo en radianes que corresponde a la tangente trinométrica cociente de ambos argumentos.
SINH	SINH DSINH	R DP	R DP	Calcula el valor del seno hiperbólico del argumento, es decir, $(e^x - e^{-x})/2$, siendo x el argumento.
COSH	COSH DCOSH	R DP	R BP	Calcula el valor del coseno hiperbólico del argumento, es decir, $(e^x + e^{-x})/2$, siendo x el argumento.
TANH	TANH DTANH	R DP	R DP	Calcula el valor de la tangente hiperbólica del argumento, es decir, $(e^x - e^{-x})/(e^x + e^{-x})$, siendo x el argumento.

NOMBRE GENERICO	NOMBRE ESPECIFICO	TIPO AGTS	TIPO FUNC	DEFINICION
	LGE	2,CH	L	Siendo x_1 y x_2 los dos argumentos, esta función devuelve el valor . TRUE . si es $x_1 = x_2$ o bien si x_1 precede x_2 en la sucesión de caracteres ordenada según la norma ASCII. En caso contrario devuelve el valor . FALSE . (Del inglés, lexically less than or equal).
	LGT	2,CH	L	Siendo x_1 y x_2 los dos argumentos, esta función devuelve el valor .TRUE. si x_1 sigue a x_2 en la sucesión de caracteres ordenada según la norma ASCII. En caso contrario devuelve el valor .FALSE. (Del Inglés, lexically greater than)
	LLE	2,CH	L	Siendo x_1 y x_2 los dos argumentos, esta función devuelve el valor . TRUE . si es $x_1 = x_2$ o bien si x_1 precede x_2 en la sucesión de caracteres ordenada según la norma ASCII. En caso contrario devuelve el valor .FALSE. (Del inglés, lexically less than or equal).
	LLT	2,CH	L	Siendo x_1 y x_2 los dos argumentos, esta función devuelve el valor .TRUE. si x_1 precede a x_2 en la sucesión de caracteres ordenada según la norma ASCII. En caso contrario devuelve el valor .FALSE. (Del Inglés, lexically less than).

En más de una ocasión en este mismo tema hemos hablado de la *ordenación de caracteres* según las normas ASCII. Veamos lo que esto significa realmente. Es obvio que tratándose de cifras, el *cero* se considera menor que el *uno* y este menor que el *dos*, etc. Este orden natural de los números es el que considera también el lenguaje FORTRAN 77. También podríamos pensar, hablando de caracteres alfabéticos, que la A precede a la B y esta a la C, etc. Pero ya no existen reglas fijas para conocer si los números preceden a las letras o cuál es la posición, en esta ordenación, que ocupa un carácter especial.

Dependiendo de la máquina o sistema que se emplee existen dos formas distintas de ordenar los caracteres alfanuméricos y especiales: una ordenación sigue la norma EBCDIC (*E*xtended *B*inary *C*oded *D*e-

cimal *I*nterchange *C*ode) mientras que la otra sigue la norma ASCII (*A*merican *S*tandard *C*ode for *I*nformation *I*nterchange).

En cada caso la ordenación respectiva de los caracteres FORTRAN es la siguiente:

ASCII	EBCDIC
Espacio	Espacio
$	.
' (comilla)	(
(	+
)	$
*	*
+	)
, (coma)	–
–	/
.	' (comilla)
/	=
0,1, ... 9	, (coma)
=	A, B, ..., Z
A, B, ..., Z	0, 1, ..., 9

Ambas ordenaciones satisfacen las necesidades del lenguaje FORTRAN 77.

Debido a la ordenación natural que siguen en ambos sistemas los números y las letras, no existen problemas al hacer comparaciones entre ellos cualquiera que sea la máquina con la que se trabaje. El problema surge cuando la comparación se efectúa entre caracteres especiales, lo cual significa que un programa que funciona en un ordenador, puede que no lo haga a satisfacción en otro, cuyos standards para la *sucesión de caracteres* no sean los mismos que los empleados en el primero.

La sucesión ordenada de caracteres recibe el nombre de *sucesión de intercalación* o *secuencia de comparación* (del Inglés, Collating Sequence), entendiendo por tal el orden o secuencia en el que aparecen los caracteres codificados dentro de una determinada máquina y que son utilizables por la misma.

Veamos a continuación algunos ejemplos sobre el uso de las funciones intrínsecas.

a) Empleando la codificación

```
X = –9
Y = ABS (X)
```

el valor almacenado en Y, sería 9.0

b) Empleando

```
X = 5.0D0
Y = DSQRT (X ** 2)
```

se obtendría para Y el valor 5.0D0

c) Con

```
X = 8.5
I = INT (X + 4)
```

se obtendría para I el valor 12 truncamiento del 12.4.

d) Con

```
CARAC = 'EL _ES'
NUM = LEN (CARAC // '_BUENO')
```

se concatena la constante 'EL_ES' con la '_BUENO', obteniendo como resultado su longitud total: 11 caracteres.

e) Con

```
MAX (I, J * 2, K + 3, 8)
```

se obtendría el valor máximo entre los contenidos de I, J * 2, K + 3 y el propio 8.

f) Suponiendo la codificación

```
CARAC = 'ESTABLECIMIENTO'
I = INDEX (CARAC, 'BLE')
J = INDEX (CARAC, 'EN')
K = INDEX (CARAC, 'E')
L = INDEX (CARAC, 'XU')
```

se obtendría como resultados:

```
I = 5  ,,  J = 12  ,,  K = 1  (la primera aparición)
L = 0  (no existe 'XU' en CARAC)
```

g) La aplicación de la función LLT así:

```
LLT ('FE', 'PA')
```

daría como resultado el valor . TRUE . ya que en secuencia ordenada de caracteres ASCII la primera constante *carácter* precede a la segunda.

h) Siendo

```
CONS = 'EXTERNO'
CARAC = 'AAA'
B = LGT (CONS (1:3), CARAC)
```

B contendría el valor . FALSE . ya que CONS (1:3) = 'EXT' y esta constante no precede a la 'AAA', de acuerdo con la sucesión ASCII.

7.5. FUNCIONES DE SENTENCIA

En el párrafo anterior 7.4 hemos ya introducido el concepto de funsión de sentencia. Vamos ahora a estudiar en profundidad esta nueva sentencia FORTRAN así como el campo de su aplicación.

Comencemos por un ejemplo. Supongamos que dentro de un programa tuviéramos, en su codificación, las sentencias siguientes:

```
      .
      .
A =  3.0 * B + C
      .
      .
D =  3.0 * E + F
      .
      .
P =  3.0 * Q + R
      .
      .
```

En este programa puede observarse que en tres ocasiones diferentes se realizan unos cálculos análogos: triplicar el valor de una variable real añadiéndole otra también real. El procedimiento opera siempre con una constante y dos variables.

De una forma análoga a como se hace en Matemáticas, podríamos definir un procedimiento o función para realizar o indicar, de un modo genérico, esta operación; por ejemplo, podría ser:

$$f(x, y) = 3x + y$$

y haciendo uso del lenguaje FORTRAN, la escribiríamos así:

```
FUNCI (X, Y) = 3.0 * X + Y
```

con lo cual habríamos definido una *función de sentencia* FORTRAN, denominada así porque se trata de una función en el sentido definido en 7.4 y porque consta de una sola sentencia. También se la conoce algunas veces con el nombre de *función fórmula.*

Una vez definida o establecida una función de sentencia como la anterior, será necesario hacer referencia a ella o llamarla, dentro del programa. Siguiendo con el ejemplo anterior, el programa FORTRAN podría quedar ahora constituido así:

```
      .
      .
FUNCI (X, Y) = 3.0 * X + Y        Definición
      .
      .
A = FUNCI (B, C)                  Llamada 1
      .
      .
D = FUNCI (E, F)                  Llamada 2
      .
      .
P = FUNCI (Q, R)                  Llamada 3
      .
      .
```

La sentencia A = FUNCI (B, C) significa "calcular el valor de A sustituyendo en la función de sentencia denominada FUNCI, y definida previamente, los *argumentos ficticios* X e Y por los *verdaderos* B y C, respectivamente". Lo mismo para D y para P, con los argumentos E, F y Q, R, respectivamente.

Los argumentos ficticios se denominan así porque son genéricos al definir la función sin tener ningún valor establecido; adquieren un valor concreto al ser sustituidos por los verdaderos o específicos de cada caso.

Una vez expuesto este ejemplo, estamos ya en condiciones de definir lo que es una función de sentencia.

Una función de sentencia es un procedimiento, indicado o definido mediante una única sentencia que es similar en forma a una sentencia de asignación aritmética, lógica o *carácter*.

Las funciones de sentencia se colocan en un programa FORTRAN después de las sentencias de especificación y en cualquier caso siempre antes de la primera sentencia ejecutable del programa. Las funciones de sentencia pertenecen a la categoría de sentencia no ejecutables.

Las funciones de sentencia responden al esquema general siguiente:

```
nombre (a1, a2, ....., an) = e
```

siendo: *nombre* el nombre simbólico con el que se designa la función de sentencia; es una variable FORTRAN y sigue, por consiguiente, las reglas de formación de estas.

a_i son nombres de variables distintas denominadas argumentos ficticios de la función de sentencia. Esta lista indica el orden, número y tipo de dichos argumentos. Deben escribirse los paréntesis aunque no exista ningún argumento en la lista. Los argumentos no pueden ser variables subdindicadas.

e es una expresión FORTRAN cualquiera válida: aritmética, lógica o *carácter*. No debe poseer ninguna variable subindicada.

Los argumentos ficticios que forman parte de la función de sentencia son unas variables independientes de las restantes de un programa;

por ello también se denominan *variables locales* o propias de la función de sentencia. Quiere esto decir que en el resto del programa pueden existir otras variables con el mismo nombre: las pertenecientes a la función de sentencia no se relacionan con ellas.

La expresión *e* del segundo miembro puede ser:

- Una constante o el nombre simbólico de una constante.
- Una variable o un elemento de un conjunto.
- La llamada a una función intrínseca.
- La llamada a otra función de sentencia previamente definida.
- La llamada a una función externa del usuario.
- Una expresión formada por todos o algunos de los elementos anteriores.

Por ejemplo:

```
ALGO (C, D) = C + 3.0 * D –E
```

representa una función de sentencia denominada ALGO que tiene dos argumentos ficticios C y D, ambos de tipo real. La expresión aritmética del segundo miembro contiene, además de los dos argumentos ficticios, una variable E y una constante 3.0.

Suponiendo que ALFA fuera el nombre de otra función de sentencia, un nuevo ejemplo, sería:

```
PRIMA (X, Y) = BETA * X + Y –ALFA (B)
```

con dos argumentos ficticios reales X e Y y la llamada a la función de sentencia ALFA entregándole el argumento verdadero B.

Ya hemos visto en el ejemplo del principio que para referenciar o llamar a una función de sentencia, es suficiente con dar su nombre y escribir entre paréntesis los argumentos verdaderos que van a sustituir a cada uno de los respectivos ficticios.

El valor resultante, una vez evaluada la función de sentencia con los argumentos verdaderos, tendrá el mismo tipo que el nombre de la función y quedará disponible para la expresión que contenía la referencia a la función de sentencia.

Por ejemplo, sea la codificación siguiente:

```
.
.
.
.
FUNC (X, Y, I) = X + Y – I
.
.
.
.
CALCU = 2.0 + FUNC (A, B, J)
.
.
.
```

El nombre de la función es FUNC y posee tres argumentos ficticios: el X, real, el Y, también real y el I que es entero. Los argumentos verdaderos se corresponden con los ficticios en número, orden y tipo. Con esta función se calcula la suma algebráica de A, B y J obteniéndose un resultado real como corresponde al nombre FUNC.

Una vez obtenido ese resultado, queda disponible para ser añadido a 2.0 y asignado el total a CALCU.

Otro ejemplo:

```
.
.
.
SUMA (A, J, B) = J ** 2 + A/B
.
.
.
DIS = SUMA (X + Y, J(2), 3.0)
```

En este caso la correspondencia entre argumentos ficticios y verdaderos es: A, X + Y; J, J(2); B, 3.0. Antes de efectuarse la transferencia a la función de sentencia, el sistema calcula las expresiones-argumentos, en este caso X + Y.

Utilizando sentencias de tipo como INTEGER, REAL, LOGICAL, etc. es posible alterar el tipo de resultado obtenido por una función de sentencia.

Por ejemplo:

```
INTEGER VELO
VELO (A) = 3.0 * A
.
.
.
.
JOTA = I + VELO (8.3)
```

calcularía el producto 3.0 * 8.3 = 24.9 transmitiendo luego únicamente el valor truncado 24 (entero) a la expresión I + VELO (8.3) = I + 24.

Como último ejemplo, también sería válido escribir:

```
FUNC (X, Y, Z) = X + Y – Z
ALFA = FUNC (A, B, FUNC (3.0, 2.0 – 7.3))
```

En este caso, la llamada primera a FUNC sirve para calcular un argumento verdadero para la segunda llamada: la más externa.

7.6. FUNCIONES EXTERNAS: SENTENCIAS FUNCTION Y RETURN

Como acabamos de ver las funciones de sentencia devuelven un sólo valor una vez ejecutadas y ellas mismas constan de una sola sentencia incorporada en el propio programa principal. Al escribir un procedimiento para realizar un cálculo puede que sea necesario emplear más de una sentencia aunque el valor de retorno o respuesta sea único. Para tales casos el lenguaje FORTRAN pone a disposición del usuario las *funciones externas,* las cuales, además, pueden devolver no sólo un único valor sino varios.

Las funciones externas tienen aún otra diferencia con las funciones de sentencia: se compilan con independencia del programa principal que las llama, por lo que todas sus *variables* son *locales* a la propia

función, o lo que es lo mismo, los nombres con los que se denominan no se relacionan con otros idénticos pertenecientes a otro módulo. Al compilarse como programa independiente, tienen que finalizar con la sentencia END.

Para emplear una función externa elaborada por el usuario, son necesarias dos etapas:

- Definición de la función externa mediante el uso de la sentencia FUNCTION como primera del módulo.
- Llamada al módulo FUNCTION definiendo los argumentos reales con los que va a trabajar dicho módulo.

Pero como en el caso de las funciones de sentencia, comencemos por un ejemplo. Supongamos que desea calcularse

$y = 0{,}5\ x + 0{,}95$ cuando sea $x \leqslant 2{,}1$

o bien

$y = 0{,}7\ x + 0{,}53$ cuando sea $x > 2{,}1$

Para ello podríamos emplear un subprograma externo FUNCTION como el siguiente:

```
    FUNCTION Y(X)
    IF(X-2.1)40,40,30
40    Y=0.5*X+0.95
    RETURN
30    Y=0.7*X+0.53
    RETURN
    END
```

Figura 7.2

Esta función externa tiene un nombre que es Y (variable real) y trabaja con un solo argumento ficticio que es el X y que será sustituido por el verdadero correspondiente cuando sea llamada desde el programa principal, citando precisamente su nombre Y.

Si el valor de esa variable X es igual o menor que 2.1 se ejecutará la sentencia número 40, que asigna un valor a Y (nombre de la FUNC-

TION) entregándose este valor al programa principal al ejecutarse la sentencia siguiente, es decir, la sentencia

```
RETURN
```

(del Inglés, retornar) y deteniéndose el proceso de la función. Si fuera X superior a 2.1 se realizarán unos cálculos similares siguiendo el camino lógico marcado por la sentencia número 30 y devolviéndose otro valor distinto al programa principal.

Obsérvese que los caminos son alternativos, por lo que el programa principal únicamente recibirá uno u otro de los valores Y: el calculado mediante la sentencia

```
Y = 0.5 * X + 0.95
```

o bien mediante la

```
Y = 0.7 * X + 0.53
```

Según esto, el esquema general de un subprograma FUNCTION será el siguiente:

```
FUNCTION nombre (a1, a2, ...., an)
      .
      .
      .
nombre =
      .
      .
      .
      .
  ai =
      .
      .
      .
RETURN
      .
      .
      .
END
```

siendo: *nombre*, el nombre simbólico del subprograma FUNCTION y que es una variable que se rige por las mismas reglas de formación que estas. El nombre es una *variable global*, es decir, conocida por todos los programas que forman parte de la programación modular efectuada para realizar un algoritmo. Debe figurar además como primer miembro de una sentencia de asignación, dentro del procedimiento.

a_i son los *argumentos ficticios*, constituidos por nombres de variables o nombres de conjuntos. La lista puede estar vacía, pero en todo caso, los paréntesis deben existir.

El subprograma debe contener al menos una sentencia RETURN y debe finalizar con la sentencia END.

Además de lo dicho en el ejemplo anterior sobre valores devueltos al programa principal, ampliaremos ahora esa información diciendo: una función externa puede tener cualquier número de argumentos ficticios y devolver al programa principal cualquier número de valores no sólo el contenido o asignado a *nombre* sino también cualquier otro asignado a uno o varios de sus argumentos a_i. Eso es lo que significa la asignación a a_i mostrada en el esquema anterior.

Por consiguiente una función externa FUNCTION puede emplearse para retornar un único valor como en el caso de las funciones de sentencia o para devolver un número arbitrario de valores en forma de argumentos de salida.

En este último caso debe tenerse en cuenta que los correspondientes argumentos verdaderos de llamada tienen que ser: una variable o un elemento de un conjunto, un conjunto o una serie de caracteres.

El tipo de los valores devueltos por un procedimiento FUNCTION al programa principal corresponderá al que tenga el nombre del módulo o los nombres de los argumentos empleados: entero, real, etc.

Sea la codificación siguiente:

```
FUNCTION A(X, Y)
A = X * Y
X = X + 1.0
Y = Y – 2.0
RETURN
END
```

este subprograma FUNCTION, denominado A, tiene dos argumentos fictios: X e Y. Se encarga de calcular el producto de ambos y asignar

ese valor a A. Al llegar a la sentencia RETURN, el proceso de este módulo externo se detendrá, entregando al programa principal los valores de A, de X (al que se ha añadido una unidad) y de Y (al que se le han quitado dos). No obstante es muy peligroso el empleo de esta técnica en la que se alteran valores de argumentos devolviéndose cambiados al programa principal. Es aconsejable no utilizarla (Véase 7.8).

Dijimos anteriormente que la segunda etapa al emplear una FUNCTION es saber cómo llamarla desde el programa principal. Ello se logra utilizando su nombre acompañado de los *argumentos verdaderos*, encerrados entre paréntesis, en la forma

```
nombre (a'1, a'2, ...., a'n)
```

y todo ello como elemento u operando de una expresión FORTRAN, siendo *nombre* el nombre del procedimiento externo FUNCTION al que se llama y a'_i los argumentos verdaderos que van a sustituir a los ficticios, por lo que deben coincidir en número, tipo y orden con estos últimos.

Los argumentos verdaderos pueden ser uno de los siguientes elementos:

- Una constante, una variable o una expresión cualquiera, excepto una expresión *carácter* que incluya la concatenación de un operando cuya especificación de longitud sea (*), un asterisco encerrado entre paréntesis.
- El nombre de un conjunto.
- El nombre de una función intrínseca.
- El nombre de otra FUNCTION.

Volvamos al subprograma A anterior llamándole desde un programa principal así:

PROGRAMA PRINCIPAL

```
B = 10.0
C = 5.0
ALF = A (B, C) + 2.0

END
```

PROCEDIMIENTO FUNCTION

```
FUNCTION  A(X, Y)
A = X * Y
X = X + 1.0
Y = Y – 2.0
RETURN
END
```

Fijándonos en el programa principal, los valores primitivos de B y C son, respectivamente, 10.0 y 5.0 y estos constituyen los argumentos verdaderos para llamar a la FUNCTION A que se corresponden en números y tipo con X e Y.

La llamada al procedimiento se efectúa justamente desde la sentencia:

```
ALF = A (B, C) + 2.0
```

en la que A(B, C) forma parte, como operando, de la expresión del segundo miembro.

El procedimiento devuelve al programa principal los siguientes valores:

```
50.0 = X * Y ≡ B * C = 10.0 * 5.0 = A
11.0 = X + 1.0 ≡ B + 1.0 = 10.0 + 1.0 = X
 3.0 = Y – 2.0 ≡ C – 2.0 = 5.0 – 2.0 = Y
```

El primero de ellos sustituirá al operando-llamada A (B, C), y los otros dos, respectivamente, a B y a C en lo que reste de ejecución del programa principal. Es responsabilidad del programador controlar estos cambios que se han producido en las variables B y C del módulo principal.

La variable ALFA tomará el valor definitivo

```
ALFA = 50.0 + 2.0 = 52.0
```

Volviendo a los argumentos ficticios, diremos que entre ellos no pueden figurar variables con subíndices. Por ello, cuando se trate de un conjunto, se escribirá únicamente su nombre, dimensionándolo además dentro del propio subprograma FUNCTION.

Por ejemplo, si X fuera un argumento ficticio, nombre de un conjunto unidimensional de 20 elementos, se escribiría

```
FUNCTION BETA (X, A, I)
DIMENSION X (20)
```

Veamos otro ejemplo. Se trata de calcular en primer lugar la media de los ocho primeros valores reales tomados de un conjunto A donde pueden existir hasta veinte. Después se calculará la media de los quince primeros y por último la media de todos ellos.

En la Figura 7.3 se muestran el programa principal y el subprograma FUNCTION, el cual será llamado desde el primero tres veces, entregándole los valores NUM = 8, NUM = 15 y NUM = 20, respectivamente.

PROGRAMA PRINCIPAL

```
PROGRAM PRINC
REAL MEDIA(3)
DIMENSION A(20)
READ *,A
NUM=8
    MEDIA(1)=RUTI(A,NUM)/NUM
NUM=15
    MEDIA(2)=RUTI(A,NUM)/NUM
NUM=20
    MEDIA(3)=RUTI(A,NUM)/NUM
STOP
END
```

SUBPROGRAMA

```
FUNCTION RUTI(X,I)
DIMENSION X(20)
SUMA=0.0
DO 1 J=1,I
1 SUMA=SUMA+X(J)
RUTI=SUMA
RETURN
END
```

Figura 7.3

Las medias respectivas se almacenarán en las variables reales MEDIA(1), MEDIA(2) y MEDIA(3) del programa principal.

En el ejemplo hemos considerado que el conjunto X poseía un máximo de 20 elementos. Si este dato no se conociera a priori, podríamos haber empleado una sentencia DIMENSION como

```
DIMENSION X (*)
```

cuyo efecto es asignar a X el mismo número de elementos que tenga su correspondiente argumento verdadero A en el programa principal.

El subprograma FUNCTION puede utilizar argumentos tipo *carácter*, ya sean variables, ya sean nombres de conjuntos. En tales casos, es preciso observar que los correspondientes argumentos verdaderos del programa de llamada, no pueden contener longitudes superiores a la de los ficticios.

En esos casos es de nuevo útil el uso del asterisco encerrado entre paréntesis, es decir, (*). Por ejemplo:

```
FUNCTION GAMA (AAA)
CHARACTER * (*) AAA
```

indicaría que el argumento ficticio tipo *carácter* AAA, tomará la longitud del correspondiente argumento verdadero, cuando, en tiempo de ejecución, el programa principal se lo transmita al subprograma.

No citar longitud, significa longitud uno. Así, por ejemplo:

```
FUNCTION GAMA (AAA)
CHARACTER AAA
```

indicaría que la longitud de AAA es de un sólo carácter.

Otros ejemplos serían:

```
CHARACTER * (*) AAA, BBB * 2
```

indicando una longitud de dos caracteres para BBB y longitud desconocida para AAA.

```
CHARACTER BBB, AAA * (*)
```

indicaría una longitud desconocida para AAA y de un sólo carácter para BBB, lo cual también podría expresarse así:

```
CHARACTER * (*) AAA, BBB * 1
```

Lo mismo que ocurría con las funciones de sentencia, estudiadas en el párrafo precedente, es posible alterar el tipo intrínseco de un subprograma FUNCTION empleando para ello sentencias de tipo como INTEGER, REAL, CHARACTER, LOGICAL y COMPLEX y con ello, a su vez, cambiar el tipo del resultado devuelto al programa principal.

Por ejemplo, escribiendo:

```
REAL FUNCTION MOMEN (X, J)
```

el valor almacenado en MOMEN y entregado al programa principal sería un valor real. El programa principal que llame a esta función deberá tener también una sentencia como:

```
REAL MOMEN
```

Cuando se emplea tipo CHARACTER, debe indicarse especificación de longitud; en caso contrario, se consideraría una longitud de uno.

Por ejemplo:

```
CHARACTER*4  FUNCTION A(X,Y)
```

significa que se trata de una función externa A que va a devolver un resultado *carácter* de longitud cuatro.

Si se emplea el caso específico del asterisco encerrado entre paréntesis, como:

```
CHARACTER * (*) FUNCTION A (X, Y)
```

ello siginificaría que la longitud del valor A devuelto, corresponderá a la longitud del último valor *carácter* asignado a A a lo largo de la ejecución del procedimiento, es decir, se trata de una longitud variable no conocida a priori:

Veamos ahora algunas reglas a tener en cuenta al escribir y utilizar los subprogramas FUNCTION:

- Un subprograma FUNCTION puede contener cualquier sentencia FORTRAN excepto las BLOCK DATA, SUBROUTINE, que serán estudiadas más adelante y PROGRAM.
- Dentro de un subprograma FUNCTION, su nombre simbólico no puede figurar en otra sentencia no ejecutable, excepto en una sentencia de tipo. En una sentencia ejecutable, el nombre puede aparecer únicamente como una variable.
- Si el tipo de la FUNCTION se indica mediante sentencia de tipo, como en los ejemplos anteriores, el nombre de la FUNCTION ya no debe aparecer en otra sentencia de tipo.

  Según estas dos normas, sería equivalente escribir:

  ```
  REAL FUNCTION MOMEN (A, B)
  ```

  o bien

  ```
  FUNCTION  MOMEN (A, B)
  REAL MOMEN
  ```

- Cuando en el procedimiento FUNCTION existe una única sentencia RETURN, su empleo puede ser sustituirlo por la END: ambas son equivalentes.

- Los nombres simbólicos de los argumentos ficticios, son variables locales para el procedimiento y por ello no pueden figurar en sentencias PARAMETER, DATA o en las aún no estudiadas EQUIVALENCE, SAVE, INTRINSIC o COMMON.
- Un argumento ficticio tipo *carácter* cuya longitud viene indicada con un asterisco entre paréntesis, no puede aparecer dentro del procedimiento como un operando del operador concatenación //.
- Un procedimiento FUNCTION puede ser llamado desde cualquier otro subprograma o procedimiento, pero no puede llamarse a sí mismo desde dentro de él (recursividad).

7.7. SENTENCIAS EXTERNAL E INTRINSIC

Existen ocasiones en que los argumentos verdaderos que pasan desde un programa principal a un programa externo no son simples variables sino, a su vez, nombres de un procedimiento externo. En este caso, el compilador necesita tener conocimiento de esta situación mediante la información adecuada que le debe proporcionar el programador.

Para ello se emplea la sentencia de especificación EXTERNAL, cuya forma general es

```
EXTERNAL a1, a2, ...., an
```

siendo los a_i nombres de argumentos verdaderos que además son nombres de programas externos. Esta sentencia debe escribirse en el programa principal de llamada.

Así, si en un módulo principal figura la sentencia

```
EXTERNAL VEL, SOC
```

dicha sentencia significaría que los nombres VEL y SOC han dejado de ser variables reales para convertirse en argumentos verdaderos y a la vez nombres de procedimientos.

Cuando los argumentos verdaderos son nombres de funciones intrínsecas, como SIN, TAN, etc., deben figurar en una lista incluida en otra sentencia de especificación que es la INTRINSIC, cuya forma general es:

```
INTRINSIC a1, a2, ...., an
```

y en la que los a_i representan nombres de funciones intrínsecas que se van a utilizar como argumentos verdaderos de otro procedimiento.

Las sentencias EXTERNAL e INTRINSIC se emplean al pasar argumentos verdaderos no solo a una función externa FUNCTION sino también a otros subprogramas que estudiaremos más adelante y que se denomirán SUBROUTINE.

La siguiente lista de funciones intrínsecas no pueden utilizarse, sin embargo, como argumentos verdaderos en una sentencia INTRINSIC:

– Funciones para conversión de tipo:

INT
IFIX
IDINT
FLOAT
SNGL
REAL
DBLE
CMPLX
ICHAR
CHAR

– Funciones de relación léxica:

LGE
LGT
LLE
LLT

– Funciones para encontrar el máximo o mínimo:

MAX
MAXO
AMAX1
DMAX1
AMAX0
MAX1
MIN
MIN0

AMIN1
DMIN1
AMIN0
MIND1

7.8. EFECTOS BORDE

Cuando se hace uso de los procedimientos FUNCTION, como ya se advirtió en 7.6, es aconsejable no emplear los argumentos ficticios como primeros miembros de sentencias de asignación para devolver así valores al programa principal. Ello puede causar muchos problemas al usuario y obtener resultados impredecibles en un determinado algoritmo.

La obtención de esos resultados erróneos al seguir la técnica descrita en el uso de una FUNCTION, recibe el nombre de *efecto de borde* (En Inglés, side effect).

Describamos esta situación con un ejemplo. Supongamos la codificación siguiente:

PROGRAMA PRINCIPAL	SUBPROGRAMA
.	FUNCTION BETA(J)
.	J = J + 1
.	BETA = J
I = 0	RETURN
ALFA = 2.0 * BETA(I)	END
.	

El argumento verdadero que se transmite a la función externa es I = J = 0.

Este valor se incrementa en una unidad y además se almacena en BETA, de modo que será:

J = 1 y BETA = 1.0

Al regresar al programa principal será:

ALFA = 2.0 * BETA(I) = 2.0 * 1 = 2.0.

Si las instrucciones del programa principal hubieran sido

```
.
.
.
I = 0
ALFA = BETA(I) + BETA(I)
.
.
```

que aparentemente son equivalentes a las del caso anterior, hubieramos obtenido un resultado completamente diferente. Por la primera llamada a BETA se hubiera obtenido BETA = 1.0 y J = I = 1. Por la segunda llamada, contenida en el segundo sumando de la expresión, como ya es J = 1, se hubiera obtenido BETA = 2.0 y J = I = 2. Con lo cual sería:

$$ALFA = BETA(I) + BETA(I) = 1.0 + 2.0 = 3.0 \neq 2.0 = 2.0 * BETA(I)$$

EJERCICIOS RESUELTOS

7.1. Encontrar los valores obtenidos al emplear las funciones intrínsecas que se indican sobre los respectivos argumentos.

a) ABS (–5) c) NINT (8.6) e) DNINT (8.6D0)
b) CABS (3.0, .0) d) ANINT (8.6) f) NINT (–3.8)

Respuesta

a) 5 c) 9 e) 9.0D0
b) 3.0 d) 9.0 f) –4

7.2. Idem que el ejercicio anterior para los casos siguientes:

a) AINT (9.2) c) MOD (6,4) e) DMOD (6.0D0, 4.0D0)
b) AINT (–9.2) d) AMOD (6.0, 4.0) f) AMOD (3.0, 1.0)

Respuesta

a) 9.0 c) 2 e) 2.0D0
b) –9.0 d) 2.0 f) 0.0

7.3. Idem que el ejercicio anterior para los casos siguientes:

a) MAX (8.0, –3.5, 2.3, 10.1, –3.0)
b) MIN (–3.2, –2.0, 2.0, 2.0, 15.2, 0.0)
c) SIGN (8,3)
d) SIGN (8.0, –3.0)

Respuesta

a) 10.1
b) –3.2
c) 8
d) –8.0

7.4. Idem que 7.1 con las funciones siguientes:

a) DIM (8,2)
b) IDIM (5,1)
c) DDIM (6.0D0, 8.0D0)
d) DIM (4,9)

Respuesta

a) 6
b) 4
c) 0.0D0
d) 0.0

7.5. Idem que los casos anteriores con:

a) INDEX ('CORRECTO', 'EC')
b) INDEX ('1AB1AB1AB', 'AB')
c) INDEX ('56A_C', '5AC')
d) INDEX ('CARBON', 'BOS')

Respuesta

a) 5
b) 2
c) 0
d) 0

7.6. Cuáles serán los valores lógicos obtenidos por las funciones que se indican teniendo en cuenta, en cada caso, el standard correspondiente.

a) LGT ('A', '8'), ASCII
b) LGT ('A', '9'), EBCDIC
e) LGE ('=', '*'), ASCII
d) LLT ('3', 'C'), EBCDIC

Respuesta

a) . TRUE .
b) . FALSE .
c) . TRUE .
d) . FALSE .

7.7. Dada la definición de función de sentencia:

SUMA (X, Y, Z) = X + Y + Z + 3.0

a la cual se llama mediante la sentencia

```
NETO = 8 + SUMA (2., -3., 4.)
```

decir qué valor será asignado a la variable NETO.

Respuesta

```
NETO = 14
```

7.8. Dada la siguiente fórmula metamática:

$$C = (r^t - 1)\, n/(r-3)$$

construir una función de sentencia equivalente.

Respuesta

```
CAPI (R, T, N) = (R**T -1.0) * N/ (R -3.0)
```

7.9. Establecer una función de sentencia para definir una de las raíces de una ecuación de segundo grado.

$$x_1 = (-b + \sqrt{b^2 - 4\,ac}) / 2a$$

Respuesta

```
RAIZ1 (A, B, C) = (-B + SQRT (B*B -4.*A*C)) / (2.*A)
```

7.10. Encontrar los errores en las siguientes definiciones de funciones de sentencia:

a) `ALFA (A, B) = A * B - ALFA (C, D)`

b)
```
A1 (X)  = 2.0 * A2 (X)
A2 (X)  = 3.0 * B + JOTA
```

c) `5AB (X) = 4.0 - X ** 2`

Respuesta

a) Se trata de una definición recursiva que no es válida. Una función de sentencia no puede definirse empleándose ella misma.

b) El orden no es válido. Debería ser

```
A2 (X) = 3.0 * B + JOTA
A1 (X) = 2.0 * A2 (X)
```

de manera que al calcular A1 (X) ya estuviera definida A2 (X).

c) El nombre de una función no puede comenzar por un carácter numérico.

7.11. Encontrar los errores en las siguientes llamadas a la función de sentencia ALFA (A, B) = 3 * A + B

a) A = 3.0 + ALFA (5.,3)
b) B = 2.0 * ALFA (5.0, 3.0, –7.2)
c) C = ALFA (I, J)

Respuesta

a) El tipo del segundo argumento verdadero no corresponde al de la definición.
b) La llamada posee más argumentos verdaderos que los que posee la función.
c) El tipo de los argumentos verdaderos no coincide con los ficticios.

7.12. Dado el procedimiento FUNCTION definido como:

```
FUNCTION  A(X)
A = EXP (X) + SQRT(X)
RETURN
END
```

y la llamada Y = 2.0 * A (B + C),
deducir el valor asignado a Y una vez ejecutada la FUNCTION.

Respuesta

Y = 2.0 * (EXP (B + C) + SQRT (B + C))

7.13. Dada la definición FUNCTION FUNCA (I, X, Y, J, Z, M) y la llamada a la misma A = FUNCA (1, P+Q, G, I, R(L), 7 * K)

contestar a las preguntas siguientes:

a) Cuáles son los argumentos ficticios.
b) Establecer la relación entre argumentos verdaderos y ficticios.
c) En qué variables del subprograma y del programa principal se encontrará el resultado obtenido por el subprograma.

Respuesta

a) I, X, Y, J, Z, M
b) 1 → I (enteros) " G → Y (reales) " R(L) → Z (reales)
P + Q → X (reales) " I → J (enteros) " 7 * K → M (enteros)
c) Primero en FUNCA y luego en A.

7.14. Escribir de otra forma la definición siguiente:

```
LOGICAL FUNCTION NON (A, B)
```

Respuesta

```
FUNCTION NON (A, B)
LOGICAL  NON
```

7.15. Dado el subprograma FUNCTION siguiente:

```
      INTEGER FUNCTION FACTO (I)
      J = I
      FACTO = 1
1     FACTO = FACTO * 5
      J = J – 1
      IF (J–1) 2,2,1
2     RETURN
      END
```

y la llamada K = FACTO (6), deducir el valor de K al final del proceso.

Respuesta

El valor de K es el factorial de esa variable, es decir, K!. En el ejemplo vale.

$K = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$

7.16. Dado el programa principal

```
CHARACTER*5 A
      .
      .
      .
C = 2.0 + ALFA (A, J + 8)
      .
      .
```

y el subprograma

```
FUNCTION ALFA (B, I)
```

Escribir las dos sentencias posibles para definir, en el sobprograma FUNCTION, la longitud del argumento B.

Respuesta

a) CHARACTER*5 B
b) CHARACTER*(*) B

8

Programación Modular: Subrutinas

Nada en exceso

ANONIMO GRIEGO

8.1. TIPOS DE LLAMADA A LOS PROCEDIMIENTOS

Vamos a iniciar este capítulo con una serie de conceptos que son aplicables no sólo a los subprogramas SUBROUTINE que estudiaremos a continuación, sino también a los procedimientos externos FUNCTION que hemos visto anteriormente.

Ya sabemos que cuando se inicia la llamada y proceso de un subprograma, tiene lugar una asociación entre los argumentos verdaderos del programa principal y los ficticios correspondientes. Para matizar sobre este tema, añadiremos ahora que la asociación puede lograrse siguiendo dos técnicas distintas: *llamada por valor* y *llamada por dirección,* conocida algunas veces como *llamada por referencia.*

En una *llamada por valor* se utilizan una copia del argumento verdadero para inicializar el ficticio, por lo cual el verdadero no cambia durante la ejecución del subprograma, incluso aunque el valor del argumento ficticio asociado varie.

En una *llamada por dirección*, la dirección del argumento verdadero se convierte en dirección del ficticio o lo que es lo mismo: es la dirección del argumento verdadero y no su valor la que pasa al subprograma, de manera que tanto argumento verdadero como ficticio se referirán ahora a la misma localización de memoria.

Esto significa además, que cualquier cambio en el valor del argumento ficticio durante el proceso del subprograma, equivale a alterar el valor del verdadero en el principal.

Los esquemas de la figura 8.1 indican ambas llamadas.

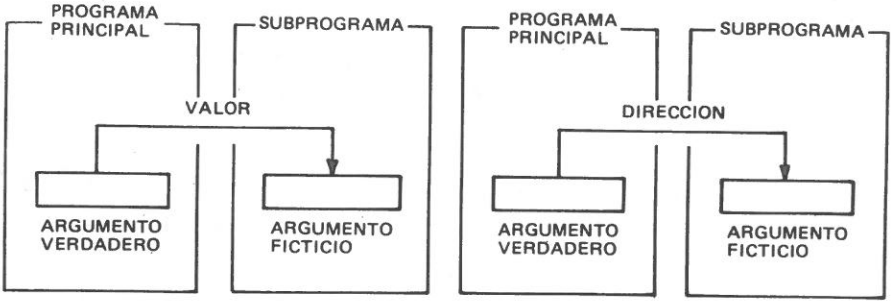


Figura 8.1

Una llamada por dirección sería la que debería emplearse cuando desde un procedimiento externo se transmitieran argumentos ficticios, además del valor contenido en el propio nombre, al programa principal. En otro caso este no tendría información sobre los mismos.

Una llamada por valor sería útil para transmitir argumentos verdaderos a un subprograma ya que de esta forma quedarían inalterados en el principal.

El lenguaje FORTRAN utiliza únicamente la técnica de llamada por dirección aunque en algunos casos, y según como se escriba el programa, puede producirse el mismo efecto que la llamada por valor.

Concretamente, las reglas utilizadas por el FORTRAN a la hora de pasar argumentos desde un programa principal a una FUNCTION o SUBROUTINE son las dos siguientes:

- Si el argumento verdadero es el nombre de una variable, el nombre de un conjunto, el elemento de un conjunto o una serie de caracteres, entonces la llamada se produce por dirección, con lo cual el valor de esos elementos enumerados puede alterarse durante la ejecución del subprograma.
- Si el argumento verdadero es cualquier otra expresión, incluyendo en esta una constante, entonces tiene efecto una llamada por valor puesto que es la única técnica que no produce cambios en el contenido de la misma durante la ejecución del subprograma.

En algunos compiladores FORTRAN existe la posibilidad de que sea el propio programador el que elija uno u otro tipo de llamada. Ello

se logra empleando los paréntesis con aquellos datos que vayan a pasar al subprograma empleando la técnica de llamada por valor. Por ejemplo, si ALFA fuera el nombre de una FUNCTION y A, B y C los argumentos verdaderos, a la hora de llamarla a ejecución, escribiendo, por ejemplo,

```
BETA = 2.0 + ALFA (A, (B), C)
```

tendría lugar una llamada por dirección para A y C (con lo cual su contenido puede modificarse) y otra por valor para B, cuyo contenido permanecería inalterado.

Así mismo tendría lugar una llamada por valor, con algunos argumentos, empleando la codificación

```
BETA = 2.0 + ALFA (5.0, 2.3, C)
```

o bien

```
BETA = 2.0 + ALFA (A, 2 * B, C)
```

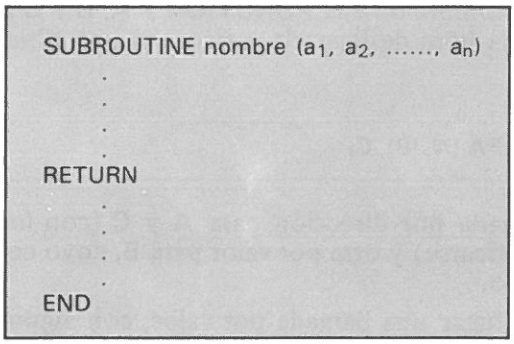
por tratarse, como hemos dicho, de las expresiones 5.0, 2.3 en el primer caso y 2*B en el segundo.

8.2. SUBRUTINAS: SENTENCIAS SUBROUTINE Y CALL

Otro de los módulos FORTRAN empleados en la estructuración de programas es la subrutina. Tales módulos son reconocibles por comenzar con la sentencia SUBROUTINE y en principio hemos de decir que tienen muchas analogías con el procedimiento FUNCTION estudiado en el capítulo precedente.

Una SUBROUTINE es un módulo que se compila independientemente del programa principal; puede poseer varias instrucciones; puede devolver al programa principal varios valores, uno sólo o quizá ninguno; emplea, con el mismo sentido y significado que en un procedimiento FUNCTION, las sentencias RETURN y END para indicar un final lógico y físico, respectivamente.

El esquema general de un subprograma SUBROUTINE es el siguiente:



siendo: *nombre*, el nombre del subprograma que es una variable FORTRAN que se rige por las reglas de éstas; a_i son los *argumentos ficticios* constituidos por nombres de variables, nombres de conjuntos (pero sin subíndices) o por un asterisco, (*). Este último tipo solamente en el FORTRAN completo.

Cuando no existen argumentos ficticios, tampoco han de escribirse los paréntesis que los contienen.

Un procedimiento SUBROUTINE sería, por ejemplo:

```
SUBROUTINE  ALFA (X, Y, I)
      .
      .
      .
Y = 0.5 * X
RETURN
END
```

En ejemplos como este, en los que únicamente existe una sentencia RETURN, las RETURN y END son equivalentes por lo que también podría haberse escrito así:

```
SUBROUTINE  ALFA (X, Y, I)
      .
      .
      .
Y = 0.5 * X
END
```

Si, como hemos dicho, la lista de argumentos ficticios falta, entonces basta con escribir la palabra SUBROUTINE seguida de su nombre.

Como puede observarse las funciones externas FUNCTION y los subprogramas SUBROUTINE difieren entre sí en algunos detalles únicamente:

- Es recomendable emplear una FUNCTION para devolver únicamente un resultado al programa principal y ese resultado es el contenido de la variable *nombre;* una SUBROUTINE produce como resultado tantos valores como sean necesarios o quizá ninguno, no empleándose su nombre para entregarlos al programa principal. Concretamente, devuelve todos aquellos valores que figurando entre sus argumentos ficticios pertenezcan al primer miembro de una sentencia de asignación contenida dentro de la subrutina. En el ejemplo anterior, el valor devuelto sería el contenido de la variable Y.
- Una FUNCTION entra en ejecución llamándola implícitamente dando su nombre y sus argumentos verdaderos; una subrutina SUBROUTINE se llama explícitamente empleando una sentencia especial CALL en el programa principal.
- Los resultados calculados en la SUBROUTINE se transmiten al programa principal no a través del nombre, si no a través de la lista de argumentos.

Como acabamos de decir, para que entre en ejecución una SUBROUTINE, es necesario escribir una sentencia CALL (del Inglés, llamar) en el programa principal. La sintaxis de esta sentencia es:

```
CALL nombre (a'1, a'2, ...., a'n)
```

siendo *nombre*, el nombre del subprograma SUBROUTINE llamado y a'_i los argumentos verdaderos que van a corresponderse con los ficticios, por lo que deben coincidir en número, tipo y orden con estos últimos.

Los argumentos verdaderos pueden ser uno cualquiera de los siguientes elementos:

- Una constante, una variable o una expresión válida FORTRAN, excepto una expresión tipo *carácter* que incluya la concatenación de un operando cuya especificación de longitud sea (*), es decir, un asterisco encerrado entre paréntesis.

- El nombre de un conjunto.
- El nombre de una función intrínseca.
- El nombre de una FUNCTION.

Para los dos últimos casos es necesario hacer uso de las sentencias INTRINSIC o EXTERNAL ya definidas en 7.7.

Analicemos el ejemplo siguiente:

PROGRAMA PRINCIPAL	SUBPROGRAMA
.	SUBROUTINE VELO (A, B, C, J)
.	.
.	.
CALL VELO (2.0, V(1), G, I)	.
.	B = A ** J
.	C = B ** J
.	.
	.
	RETURN
	END

El nombre de la subrutina es VELO, siendo 2.0, V(1), G e I los argumentos verdaderos que se relacionan, respectivamente con A, B, C y J. Los valores transmitidos al programa principal son los contenidos en B y C (argumentos ficticios y a la vez primeros miembros de una sentencia de asignación).

Esos valores se encontrarán en el programa principal, en las variables V (1) y G, respectivamente.

Otro ejemplo sería el contenido en la figura 8.2.

PROGRAMA PRINCIPAL	SUBPROGRAMA
PROGRAM AREA	
DIMENSION BASE(10),ALTU(10),SUPER(10)	SUBROUTINE AREA (B,A,S)
.	S=B*A/2.0
.	RETURN
.	END
DO 3 I=1,10	
3 CALL AREA (BASE(I),ALTU(I),SUPER(I))	

Figura 8.2

En este ejemplo puede observarse que el programa principal en un determinado momento de su ejecución, al aparecer la sentencia CALL AREA, transfiere el control a la subrutina AREA, transmitiéndole los argumentos verdaderos BASE (I), ALTU (I) y SUPER (I) mediante una llamada por dirección. Estos argumentos corresponden, respectivamente, a la base, altura y superficie de un triángulo cualquiera. Los argumentos ficticios son B, A y S. El subprograma calcula el área del triángulo almacenando el resultado en S y transmitiendo este valor al programa principal, que se almacenará en SUPER (I).

Realmente los argumentos que el programa principal entrega al subprograma son BASE (I) y ALTU (I). El valor que el subprograma devuelve es SUPER (I).

8.3. UTILIZACION DE CONJUNTOS EN FUNCIONES Y SUBRUTINAS

Es importante analizar cómo es la correspondencia entre argumentos ficticios y verdaderos cuando se trata de conjuntos de valores o datos. Lo que vamos a exponer a continuación sirve tanto para el caso de una FUNCTION como de una SUBROUTINE.

Haremos el análisis proponiendo varios ejemplos, pero como regla general no debe olvidarse que los argumentos ficticios pueden ser nombres de conjuntos sin subíndice, mientras que los correspondientes argumentos verdaderos pueden ser también nombres de conjuntos o nombres de elementos de un conjunto.

a) Supongamos en primer lugar la codificación siguiente:

PROGRAMA PRINCIPAL	SUBPROGRAMA
INTEGER BETA (3:20)	SUBROUTINE VELO (ALFA)
.	INTEGER ALFA (1:18)
.	.
CALL VELO (BETA)	.
	.
	.

Entonces la asociación o correspondencia de valores entre el programa principal y el subprograma, es así:

UBICACION	CONJUNTO	
	BETA	ALFA
1	BETA (3)	ALFA (1)
2	BETA (4)	ALFA (2)
3	BETA (5)	ALFA (3)
.	.	.
.	.	.
.	.	.
17	BETA (19)	ALFA (17)
18	BETA (20)	ALFA (18)

Es decir, la asociación se realiza *por dirección* de manera que el primer elemento de BETA se corresponde con el primero de ALFA y así sucesivamente hasta finalizar uno u otro conjunto.

b) Cuando se escribe la subrutina es posible que no se conozca el valor máximo de una dimensión. Entonces puede emplearse el asterisco, *, para indicarlo, pero únicamente tratándose de argumentos ficticios.

Por ejemplo, dada la codificación

```
PROGRAMA PRINCIPAL          SUBPROGRAMA
INTEGER BETA (-1:10)        FUNCTION VELO (ALFA)
  .                         INTEGER ALFA (2:*)
  .                           .
  .                           .
A = 2.0 * VELO (BETA)         .
```

la asociación sería:

UBICACION	CONJUNTO	
	BETA	ALFA
1	BETA (-1)	ALFA (2)
2	BETA (0)	ALFA (3)
3	BETA (1)	ALFA (4)
.	.	.
.	.	.
.	.	.
.	.	.
11	BETA (9)	ALFA (12)
12	BETA (10)	ALFA (13)

es decir, el número de elementos de ambos conjuntos coincide.

Un conjunto argumento ficticio no puede tener nunca más elementos que los que posee el conjunto de argumentos verdaderos.

Lo dicho en estos ejemplos es extensible al caso de argumentos tipo *carácter* cuando las longitudes de cada elemento del conjunto ficticio y verdadero coinciden.

c) Hemos repetido varias veces, que la correspondencia entre argumentos verdaderos y ficticios tiene que ser en número, orden y tipo. Tratándose sin embargo de argumentos conjuntos tipo *carácter*, la longitud no tiene por qué coincidir.

Sea la codificación siguiente:

PROGRAMA PRINCIPAL	SUBPROGRAMA
CHARACTER BETA (1:6)*10	SUBROUTINE VELO (ALFA) CHARACTER ALFA (2:*) * 5
.	.
.	.
.	.
CALL VELO (BETA)	

Entonces la asociación sería ahora la siguiente:

		CONJUNTO		
UBICACION	BETA	CARAC.	ALFA	CARAC.
1	BETA (1)	10	ALFA (2)	5
			ALFA (3)	5
2	BETA (2)	10	ALFA (4)	5
			ALFA (5)	5
3	BETA (3)	10	ALFA (6)	5
.	.		ALFA (7)	5
.	.		.	
.	.		.	
.	.		.	
6	BETA (6)	10	ALFA (12)	5
			ALFA (13)	5

La única limitación en este caso es que el número total de caracteres del conjunto ficticio no puede superar al número total de caracteres del conjunto verdadero.

Si en la subrutina se hubiera indicado además una longitud desconocida para cada variable tipo *carácter* del conjunto ALFA, es decir, si se hubiera escrito

```
CHARACTER ALFA (2:*) * (*)
```

entonces el número de caracteres de cada argumento ficticio coincidiría con el número de caracteres del argumento verdadero correspondiente y si los argumentos son conjuntos, como es el caso del ejemplo, entonces la regla se aplica a los tamaños totales de ambos conjuntos.

d) Veamos ahora un nuevo ejemplo pero asociando un elemento de un conjunto como argumento verdadero con un conjunto completo como argumentos ficticios.

Sea por ejemplo, la codificación:

PROGRAMA PRINCIPAL	SUBPROGRAMA
INTEGER BETA (1:5)	SUBROUTINE VELO (ALFA)
.	INTEGER ALFA (1:*)
.	.
.	.
.	.
CALL VELO (BETA (3))	

entonces esta sería la correspondencia:

	CONJUNTO	
UBICACION	BETA	ALFA
1	BETA (1)	
2	BETA (2)	
3	BETA (3)	ALFA (1)
4	BETA (4)	ALFA (2)
5	BETA (5)	ALFA (3)

En este caso la dirección de BETA (3) pasa al subprograma y esa dirección también será la del primer elemento del conjunto ficticio ALFA. De la misma forma BETA (4) coincidirá con ALFA (2) y así sucesivamente hasta alcanzar el final del conjunto verdadero BETA.

8.4. DIMENSIONES AJUSTABLES

El lenguaje FORTRAN dispone de otra posibilidad para lograr una correspondencia entre argumentos verdaderos y ficticios cuando se trata de conjuntos y se desconoce el número máximo de elementos que posee el ficticio en tiempo de compilación, resolviéndose sin embargo dicho problema en tiempo de ejecución.

Siguiendo esta facilidad, se pueden declarar en subprogramas y sólo en ellos, conjuntos cuyas dimensiones vengan expresadas por *variables enteras.*

Por ejemplo, si tuvieramos:

PROGRAMA PRINCIPAL	SUBPROGRAMA
DIMENSION BETA (10)	SUBROUTINE VELO (ALFA, I)
.	DIMENSION ALFA (I)
.	.
.	.
.	.
CALL VELO (BETA, 3)	
.	
.	
.	
.	
CALL VELO (BETA (4), 6)	
.	
.	
.	

entonces el número máximo de elementos del conjunto ALFA de la subrutina está indicado por una variable entera I, que corresponde a la constante 3 en la primera llamada y a la 6 en la segunda, de manera que, en tiempo de ejecución, el subprograma conocerá concretamente cual es el valor máximo del subíndice del conjunto ALFA.

La correspondencia entre variables será la siguiente:

UBICACION	BETA	ALFA (llamada 1)	ALFA (llamada 2)
1	BETA (1)	ALFA (1)	
2	BETA (2)	ALFA (2)	
3	BETA (3)	ALFA (3)	
4	BETA (4)		ALFA (1)
5	BETA (5)		ALFA (2)
6	BETA (6)		ALFA (3)
.	.		
.	.		
.	.		
9	BETA (9)		ALFA (6)
10	BETA(10)		
.	.		
.	.		
.	.		
12	BETA(12)		

Todo ello coincide, no obstante, con las reglas dadas en el párrafo precedente.

8.5. ENTRADAS Y SALIDAS MULTIPLES: SENTENCIA ENTRY

En todo lo que hemos visto hasta aquí, los procedimientos FUNCTION y SUBROUTINE empezaban a ejecutarse partiendo de la primera sentencia cabecero que se encontraba en uno u otro procedimiento. Pero está no es la única forma de llamar al subprograma. Una nueva sentencia, la sentencia ENTRY, permite que un determinado procedimiento comience en otra sentencia específica contenida en una función externa o subrutina.

La sentencia ENTRY puede aparecer en cualquier lugar de una FUNCTION o de una SUBROUTINE, excepto dentro de un bloque IF, END IF o bien dentro de un bucle DO.

La sentencia ENTRY, que no figura en el FORTRAN básico, no es ejecutable y pueden existir una o varias dentro de un mismo procedimiento.

La forma general de esta sentencia es

```
ENTRY nombre (a1, a2, ....., an)
```

Como puede observarse, la palabra ENTRY ha reemplazado a las FUNCTION o SUBROUTINE. Por consiguiente, tanto *nombre*, como a_i, tienen aquí el mismo significado que tenían en los procedimientos citados.

Los nombres correspondiente a varios puntos de entrada tienen que ser diferentes entre sí y diferentes del nombre de la propia FUNCTION o SUBROUTINE donde esté contenida la sentencia ENTRY.

La lista de argumentos ficticios a_i, puede ser diferente para cada punto de entrada.

A un punto de entrada ENTRY se accede desde un programa principal citando simplemente su nombre (caso de una FUNCTION) o llamándolo mediante una sentencia CALL (caso de una SUBROUTINE).

Por ejemplo, sea la codificación siguiente:

```
PROGRAMA PRINCIPAL          SUBPROGRAMA

    .                       SUBROUTINE ALFA (AA, BB, CC, DD)
    .                           .
    .                           .
    .                           .
CALL ALFA (A, B, C, D)          .
    .                       ENTRY PUN1 (II, JJ)
    .                           .
    .                           .
CALL PUN1 (I, J)                .
    .                       ENTRY PUN2 (YY, ZZ)
    .                           .
    .                           .
CALL PUN2 (Y, Z)                .
                            RETURN
                            END
```

En este ejemplo, la primera llamada tiene lugar al comienzo de la subrutina ALFA entregándole los argumentos verdaderos A, B, C y D que reemplazarán respectivamente a los AA, BB, CC y DD.

La segunda llamada se produce directamente al ENTRY PUN1 sustituyendo I, J a II, JJ, respectivamente.

La última llamada a ALFA se hace directamente al ENTRY PUN2 entregándose los argumentos Y, Z que sustituirán a los YY y ZZ.

En todos los casos el proceso del subprograma prosigue hasta encontrar la primera salida RETURN.

De la misma forma que un subprograma puede poseer varios puntos de entrada, es también posible devolver el control desde una SUBROUTINE a un programa principal, a un punto distinto de la sentencia siguiente a la sentencia CALL.

Esto se logra empleando la sentencia RETURN especial, únicamente existente en el FORTRAN completo, cuya forma general es

```
RETURN e
```

siendo *e* una expresión entera FORTRAN.

El subprograma SUBROUTINE puede entonces regresar a cualquier sentencia del programa principal que posea una etiqueta. Las etiquetas de regreso han de transmitirse como argumentos verdaderos en una sentencia CALL al subprograma y en la forma

```
* etiqueta
```

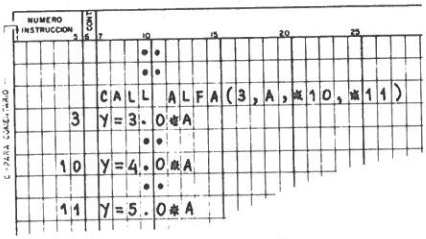
Los argumentos ficticios asociados en el procedimiento serán simplemente asteriscos *.

El valor de *e* en la sentencia RETURN tiene que ser positivo distinto de cero y no mayor que el número total de asteriscos que figuren en la sentencia SOBROUTINE del procedimiento.

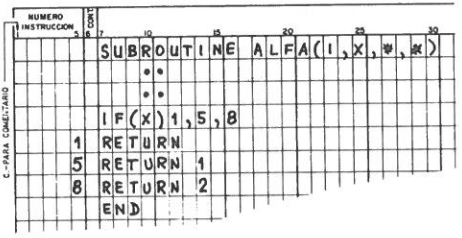
Como regla general diremos que tanto los puntos ENTRY como los retornos variables, no son recomendables en la programación.

Un ejemplo de retorno variable podría ser el indicado en la codificación de la página siguiente, donde se representan varias sentencias escritas correspondientes a un programa principal que llama a una subrutina ALFA.

PROGRAMA PRINCIPAL



SUBPROGRAMA



En la sentencia CALL del programa principal, los argumentos verdaderos *10 y *11, hacen relación a las etiquetes respectivas 10 y 11. Cuando el subprograma transcurra por el camino lógico número 1 de la sentencia IF, se realizará su retorno normal a la sentecia número 3 del programa principal (sentencia siguiente a la CALL). Cuando el camino lógico sea el 5, el retorno se hará a la sentencia cuya etiqueta viene indicada por el primer argumento ficticio *, que es el *10 verdadero, es decir, etiqueta número 10 del programa principal. Finalmente si el camino es el 8, el retorno será el señalado por el segundo *, es decir, á la sentencia número 11.

8.6. SENTENCIA SAVE

Todas las variables que componen una SUBROUTINE o una FUNCTION son *variables locales* al propio procedimiento. Pues bien; normalmente, y una vez que el proceso de la subrutina o función ha finalizado, todos sus conjuntos y variables locales desaparecen al control del programador de manera que si el procedimiento comenzase una vez más su proceso, este no "recordaría" cuales eran los valores anteriores que poseían sus variables.

Si el programador necesita que entre dos procesos distintos de una misma SUBROUTINE, por ejemplo, se conserve el valor de una o varias variables, de manera que una ejecución "recuerde" el valor que tenía una variable en la ejecución previa, puede emplear la sentencia SAVE, cuya forma general es

```
SAVE (lista de variables)
```

en la cual *lista de variables* significa todas aquellas variables locales al procedimiento cuyos valores quieren guardarse para el proceso siguiente. En la lista de variables puede incluirse: el nombre de una variable, el nombre de un conjunto o el nombre de un bloque COMMON con etiqueta que será estudiado posteriormente (8.10).

Una sentencia SAVE que no contenga la lista de variables entre paréntesis, equivaldría a una SAVE en la que fueran a retenerse lo valores de todas las variables que compongan la FUNCTION o la SUBROUTINE.

Las variables cuyos valores se retienen mediante el empleo de la sentencia de especificación SAVE, se las suele denominar *variables estáticas.*

Por ejemplo, si tuvieramos la codificación

```
SUBROUTINE ALFA (A, B)
SAVE I
DATA I/1/
I = I + 1
  .
  .
  .
```

Significaría que a la variable entera I se le asigna el valor 1 mediante la sentencia DATA convirtiéndose luego en 2 por la sentencia I = I + 1. Ese valor se guardará para un proceso posterior de ALFA, de modo que al finalizar éste, tomará el nuevo valor 3 debido a la sentencia de asignación aritmética e ignorando la existencia de la sentencia DATA en esta nueva ocasión.

8.7. SENTENCIA EQUIVALENCE

En los programas de cualquier tipo que tengan una longitud considerable, puede resultar conveniente tratar de ahorrar memoria de ordenador. Actualmente, esto no supone graves problemas ya que las máquinas modernas disponen de memoria amplia y además ésta es barata. No obstante y como veremos enseguida, las sentencias de que dispone el lenguaje FORTRAN para economizar memoria provienen de versiones anteriores del lenguaje y cumplen además otros objetivos y misiones. Estas sentencias son EQUIVALENCE y COMMON. La primera de ellas se utiliza para conseguir una economía de memoria dentrozo de un mismo y único módulo; con la segunda, el ahorro de me-

moria se realiza entre un programa o módulo principal y un subprograma o bien entre dos módulos distintos que sean subprogramas.

Veamos un primer caso en el que el uso de la sentencia EQUIVALENCE es conveniente o práctico. Un programa cualquiera puede contener variables que se utilicen en momentos diferentes dentro de su algoritmo pero nunca simultáneamente; tales variables, aunque de nombres diferentes, podrían estar contenidas en la misma ubicación de la memoria, lo que supondría un ahorro de esta. Esto se logra con el empleo de la sentencia EQUIVALENCE. Si esas variables fueran A y B, se escribiría.

```
EQUIVALENCE (A, B)
```

Otra situación podría ser la siguiente. Supongamos que al comenzar a escribir un programa, una determinada variable se ha denominado con un cierto nombre y luego se decide cambiar su denominación. Ambos nombres representan una misma cosa, es decir, serán equivalentes y pueden ocupar por consiguiente la misma ubicación dentro de la memoria del ordenador porque se están refiriendo al mismo valor.

Si esas denominaciones distintas de un mismo concepto o dato son X e Y, escribiríamos.

```
EQUIVALENCE (X, Y)
```

Entonces una referencia a la variable X es lo mismo que una referencia a la variable Y.

Concretando, la forma general de una sentencia EQUIVALENCE es

```
EQUIVALENCE (a1, a2, ..., an), (b1, b2, ..., bm), ...
```

siendo a_i, b_i,... nombres distintos de una misma variable o dato. Pueden ser nombres de variables, nombres de elementos de un conjunto, nombres de conjuntos y nombres de series de caracteres. Cada lista entre paréntesis debe contener al menos dos nombres.

La coma (,) que separa cada una de las listas es opcional.

Dos reglas han de observarse:

– En las listas no pueden aparecer nombres de variables que a su vez sean nombres de funciones.

– Toda expresión subíndice o expresión serie de caracteres que aparezca en la lista deberá constituir una expresión constante entera.

Supongamos la sentencia:

```
EQUIVALENCE (A, B), (X, Y, Z)
```

Esta indica que, las variables A y B comparten la misma ubicación. Lo mismo ocurre con las X, Y y Z. Pero la ubicación de estas últimas es distinta de la ubicación de las primeras.

Otro ejemplo: la sentencia

```
EQUIVALENCE (VELO (3), X), (X, ALFA)
```

sería equivalente a esta otra:

```
EQUIVALENCE (VELO (3), X, ALFA)
```

Como ya sabemos, los conjuntos se almacenan en memoria siguiendo un orden secuencial. Teniendo en cuenta esto, cuando se hacen equivalentes dos elementos de dos conjuntos diferentes, de forma automática se hacen equivalentes otros elementos, o quizás todos, de ambos conjuntos. Así, el empleo simultáneo de las dos sentencias

```
DIMENSION A(4), B(7)
EQUIVALENCE (A(2), B(4))
```

proporcionaría un esquema de memoria como éste:

	CONJUNTO	
UBICACION	A	B
1		B(1)
2		B(2)
3	A(1)	B(3)
4	A(2) ⇔	B(4)
5	A(3)	B(5)
6	A(4)	B(6)
7		B(7)

Es decir A(1) se refiere al mismo dato que B(3); A(2) al mismo que B(4), etc.

Cuando se trate de conjuntos de dos dimensiones, por ejemplo, y recordando que estos se almacenan por columnas, o lo que es lo mismo, variando más rápidamente el segundo subíndice, tendríamos un esquema de memoria como el que se indica a continuación, al escribir las dos sentencias siguientes:

```
DIMENSION A(5), B(2, 3)
EQUIVALENCE (A(1), B(2,1))
```

UBICACION	CONJUNTO A	CONJUNTO B
1		B(1,1)
2	A(1) ⇔	B(2,1)
3	A(2)	B(1,2)
4	A(3)	B(2,2)
5	A(4)	B(1,3)
6	A(5)	B(2,3)

Las dos sentencias anteriores podrían haberse escrito también en la forma

```
DIMENSION    A(5), B(2,3)
EQUIVALENCE  (A(1), B(2))
```

En este caso B(2) significa el segundo elemento del conjunto bidimensional B.

En una sentencia EQUIVALENCE no pueden existir incompatibilidades. Así por ejemplo, no sería lícito escribir.

```
EQUIVALENCE (A(1), B(2)), (A(1), C), (C, B(3))
```

ya que se harían equivalentes, en primer lugar, A(1) con B(2) y luego A(1) con B(3) por intermedio de A(1) y C. En definitiva, dos ubicaciones diferentes para una misma variable, la A(1).

El hecho de hacer equivalentes entre sí variables de diferentes tipo no quiere ello decir que se produzca un cambio de tipo en ninguna de ellas. De la misma forma, cuando una varible y un conjunto se hacen

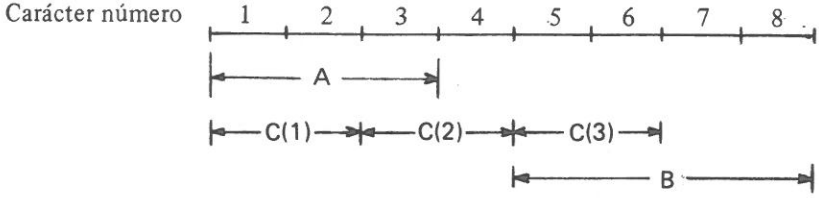
equivalentes, la variable no tendrá las propiedades del conjunto ni este las de aquella.

Los datos de tipo *carácter* únicamente pueden hacerse equivalentes a otros datos de tipo *carácter*, aunque sus longitudes no tengan que ser las mismas. La única observación a tener en cuenta es que los datos al hacerse equivalentes tendrán su primer carácter ubicado en la misma posición de memoria.

Por ejemplo, sea la codificación:

```
CHARACTER  A*3, B*4, C(3)*2
EQUIVALENCE (A, C(1)), (B, C(3))
```

entonces el esquema de equivalencias podría ser el siguiente:



es decir, el primer carácter de A ocuparía la misma ubicación que el primer carácter de C(1); análogamente, el primer carácter de B tendría la misma ubicación que el primero de C(3). Las demás equivalencias son una consecuencia de la propia codificación escrita.

Por último escribamos ahora un ejemplo de equivalencia entre una variable y el nombre de un conjunto:

```
DIMENSION A(3)
EQUIVALENCE (A, B)
```

Entonces la correspondencia sería:

UBICACION	CONJUNTO A
1	A(1) ⇔ B
2	A(2)
3	A(3)

es decir, B con el primer elemento del conjunto A.

8.8. SENTENCIA COMMON

Ya se ha dicho en el párrafo precedente que el ahorro de memoria puede también conseguirse cuando se trata de un programa principal y un subprograma, es decir, definiendo una *zona* de memoria *común* a dos módulos diferentes, mediante el empleo de la sentencia COMMON.

Veamos un primer ejemplo. Sea la codificación siguiente:

PROGRAMA PRINCIPAL	SUBPROGRAMA
COMMON A, B	SUBROUTINE SUM (K, L)
.	COMMON X, Y
.	.
.	.
CALL SUM (I, J)	.
.	
.	
.	

Las sentencias COMMON A, B y COMMON X, Y pertenecientes, respectivamente, al programa principal y al subprograma, proporcionan un área común para las variables A y X y otra para las variables B e Y, estando unas y otras situadas en programas diferentes.

Otra aplicación de la sentencia COMMON, quizás la más usual, es la de poder transmitir a un subprograma los argumentos verdaderos que le envía un programa principal y que van a sustituir a los ficticios. Hasta ahora, por ejemplo, hemos visto que el enlace entre un programa principal y una SUBROUTINE se hacía mediante el empleo de una sentencia CALL y los argumentos verdaderos encerrados entre paréntesis, como

```
CALL ALFA (X, Y, Z)
```

y utilizando una sentencia cabecero en el subprograma, como por ejemplo:

```
SUBROUTINE ALFA (R, S, T)
```

Haciendo uso de la sentencia COMMON conseguiríamos el mismo efecto codificando así:

PROGRAMA PRINCIPAL	SUBPROGRAMA
COMMON X, Y, Z	SUBROUTINE ALFA
.	COMMON R, S, T
.	.
.	.
CALL ALFA	.
.	
.	
.	

El resultado es que los argumentos verdaderos y ficticios se han escrito en una sentencia COMMON, en el programa principal y en el subprograma, y a través de la zona común de la memoria así definida, se transmiten de un programa a otro. El resultado obtenido, al utilizar esta técnica para subrutinas y funciones externas, es que el programa FORTRAN es más eficiente tanto en cuanto a memoria empleada, como a tiempo de ejecución gastado por el ordenador.

Esa zona común la esquematizaremos así:

UBICACION	1	2	3
VARIABLES	X	Y	Z
ZONA COMUN	R	S	T

Obsérvese que la zona común es posible porque a la hora de ejecución de un programa, residen en memoria tanto el módulo principal que llama, como los módulos subprograma que son llamados.

La forma general de una sentencia COMMON es:

```
COMMON a1, a2, ...., an
```

siendo a_i: nombres de variables o nombres de conjuntos que a su vez pueden contener información sobre sus dimensiones.

Como observación general diremos que las variables que figuren en la sentencia COMMON de un subprograma no pueden, a su vez, figurar entre la lista de argumentos ficticios del mismo.

Por ejemplo:

```
DIMENSION A(4)
COMMON B, A, C
```

sería equivalente a escribir:

```
COMMON B, A(4), C
```

y según lo dicho, no sería válido escribir

```
SUBROUTINE ALFA (A, B, C, D)
COMMON A, X, Y
```

ya que la variable A, que es un argumento ficticio, figura además en la lista de la zona común.

Otro ejemplo. Escribir

```
COMMON A, B, C
COMMON X, Y
```

es equivalente a escribir únicamente

```
COMMON A, B, C, X, Y
```

En cualquier caso, el sistema reserva ubicaciones en la zona común para cada variable que aparece en una sentencia o sentencias COMMON y justamente en el mismo orden en que aparecen aquellas. En el ejemplo anterior, primero para la A y a continuación para B, C, X e Y, cualquiera que sea el número de sentencias que figuren en el programa principal o subprograma.

Un nuevo ejemplo. Supongamos la sentencia

```
COMMON A, B(5), C
```

de un programa principal, y la

```
COMMON D(6), E
```

de un subprograma. Entonces la zona común creada, sería:

UBICACION	1	2	3	4	5	6	7
VARIABLES ZONA COMUN	A	B(1)	B(2)	B(3)	B(4)	B(5)	C
	D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	E

Obsérvese que si una zona común contiene una variable o un conjunto tipo *carácter*, el resto de las variables de esa zona deberán ser también tipo *carácter*.

8.9. UTILIZACION CONJUNTA DE SENTENCIAS COMMON Y EQUIVALENCE

Es muy común en los programas FORTRAN hacer equivalentes entre sí dos variables, una de las cuales reside en una zona común. Cuando eso sucede, la primera de ellas también quedará incluida en la zona común.

Vamos a mostrar esta técnica proponiendo dos ejemplos, uno de los cuales es válido y en el otro se indicará dónde no está conforme con la regla a seguir.

Sean las sentencias de un programa principal las siguientes:

```
DIMENSION X(6)
COMMON A, B(2), C(3)
EQUIVALENCE (C(1), X(3))
```

La segunda de ellas, por sí sola, crearía la siguiente zona común:

UBICACION	1	2	3	4	5	6
VARIABLES ZONA COMUN	A	B(1)	B(2)	C(1)	C(2)	C(3)

Ahora bien; al hacer equivalentes entre sí C (1) con X(3), el conjunto X pasará completo a pertenecer a la zona común, quedando ésta con la siguiente estructura:

UBICACION	1	2	3	4	5	6	7
VARIABLES	A	B(1)	B(2)	C(1)	C(2)	C(3)	
ZONA COMUN		X(1)	X(2)	X(3)	X(4)	X(5)	X(6)

En este caso, la longitud del COMMON definitivo será de siete variables, habiéndose ampliado el primitivo en una variable más, por su parte final, lo cual es correcto.

No sería correcto, sin embargo tratar de ampliar la zona común por su parte inicial. Veamos la codificación siguiente:

```
DIMENSION X(6)
COMMON A, B(2), C(3)
EQUIVALENCE (C(1), X(5))
```

Entonces la zona común tendría el esquema:

UBICACION		1	2	3	4	5	6
VARIABLES		A	B(1)	B(2)	C(1)	C(2)	C(3)
ZONA COMUN	X(1)	X(2)	X(3)	X(4)	↕ X(5)	X(6)	

el cual indica que no existe ubicación para la variable X(1). La codificación anterior no sería válida.

Por último diremos que dos variables definidas en el COMMON no pueden a su vez hacerse equivalentes. Así, no sería correcto escribir:

```
COMMON A(3), B, C
EQUIVALENCE (A(1), B)
```

ya que la variable B ocuparía su propia posición dentro de la zona común (ubicación 4) y además la de A(1) (ubicación 1).

8.10. SENTENCIA COMMON CON USO DE ETIQUETAS

La sentencia COMMON, tal como se ha definido anteriormente, crea una sola zona común de memoria, a la cual accederán dos módu-

los distintos compartiendo memoria, como ya hemos señalado, y todas las variables en ella definidas.

Puede resultar conveniente distintas zonas comunes, cada una de ellas identificada con un nombre o etiqueta, con lo cual se conseguiría que un determinado módulo tuviera acceso únicamente a algunas de las variables contenidas en la zona común, pero no a todas ellas. De esa manera el programador dispondrá de una técnica adecuada con la cual diferentes subconjuntos de variables serán compartidas por diferentes programas.

Este efecto se consigue en FORTRAN empleando la sentencia COMMON con etiqueta, también denominada, algunas veces, *con nombre*. Un COMMON se dice que es con etiqueta cuando se escribe su nombre entre dos *slashes* precediendo a todas las variables que contiene. Así, por ejemplo:

```
COMMON / BL1 /  A, B, C, D(20)
```

define una zona común cuyo nombre es BL1 que contiene las variables A, B, C y un conjunto unidimensional D de veinte elementos.

Aunque ello no es aconsejable, en una misma sentencia COMMON pueden definirse dos o más áreas comunes mediante el uso de dos o más etiquetas.

Por ejemplo:

```
COMMON / BL1 / A, B, C, D(20), / BL2 / E, F, G
```

define las áreas comunes BL1 y BL2, la primera con veintitrés variables y la segunda con únicamente tres.

Obsérvese que la coma que precede a la etiqueta del segundo bloque o zona común, es opcional.

Debido a que el COMMON definido en primer lugar no poseía etiquetas, es por lo que también se le conoce con el nombre de COMMON *blanco*, aunque dicho COMMON también se puede incluir ahora en una sentencia como la que acabamos de presentar, escribiendo únicamente dos slashes.

Así, por ejemplo:

```
COMMON / BL1 / A, B, C // X, Y, Z, V / BL2 / D, E(5)
```

define tres diferentes zonas comunes: la BL1 con tres varibles, la BL2

con seis, ambas con etiqueta y una zona común blanca con cuatro varaibles X, Y, Z y V.

Veamos ahora un ejemplo de como utilizar el COMMON con etiquetas dentro de módulos diferentes de programación.

Sea un programa principal que tuviera las sentencias:

```
COMMON X, Y
COMMON / BL1 / A1, A2 / BL2 / B1, B2, B3
```

y dos subprogramas, tales que en el primero de ellos existiera la siguiente zona común con etiqueta:

```
COMMON / BL1 / R, S
```

mientras que en el segundo se hubiera escrito

```
COMMON / BL2 / U, V, W
```

Todo ello lo podríamos esquematizar de la forma siguiente:

UBICACION	1	2
ZONA COMUN BLANCA	X	Y

UBICACION	1	2
ZONA COMUN BL1	A1	A2
	R	S

UBICACION	1	2	3
ZONA COMUN BL2	B1	B2	B3
	U	V	W

De este modo las variables A1 y A2 serán conocidas para el programa principal y únicamente para el primero de los subprogramas; igual-

mente, las variables B1, B2 y B3 únicamente serán compartidas por el programa principal y el segundo de los subprogramas.

Obsérvese que la codificación en el programa principal, también podría haber sido

```
COMMON // X, Y / BL1 / A1, A2 / BL2 / B1, B2, B3
```

indicando un COMMON blanco.

Haciendo uso del COMMON con etiquetas pueden incluirse en la misma sentencia variables de cualquier tipo, *carácter* o numéricas; ahora bien unas y otras deben pertenecer a etiquetas diferentes de modo que la mezcla en mismo COMMON con nombre sigue estando prohibida.

Las variables lógicas pueden compartir la misma zona etiquetada que las numéricas pero nunca con variables tipo *carácter.*

Tanto el COMMON blanco como el COMMON con etiquetas tienen las mismas propiedades, pero es necesario que resaltemos tres diferencias:

- Los COMMON con etiquetas con el mismo nombre, tienen que tener el mismo tamaño en todos los módulos donde aparezcan; el COMMON blanco puede tener diferentes tamaños. Así, no sería lícito escribir

  ```
  COMMON / AAA / A, B, C
  ```

  en un módulo, y

  ```
  COMMON / AAA / X, Y
  ```

  en otro

- Los datos contenidos en un COMMON con etiquetas pueden ser inicializados con un valor por medio de la sentencia BLOCK DATA que estudiaremos a continuación; los datos pertenecientes a un COMMON blanco no pueden inicializarse.
- La ejecución de las sentencias RETURN o END de un subprograma que además posee un COMMON con etiquetas pueden convertir en indefinidas (de valor desconocido) algunas variables; eso nunca ocurrirá con el uso del COMMON blanco.

8.11. SUBPROGRAMA BLOCK DATA

La sentencia BLOCK DATA es realmente la primera sentencia de un nuevo subprograma cuyo objetivo vamos a exponer a continuación.

Supongamos que tuvieramos la codificación

```
DIMENSION A(10), B(5)
COMMON /BL1/ A, I, J /BL2 / B
```

Si desearamos inicializar a *ceros* todas la variables de los conjuntos A y B, con lo que sabemos hasta aquí deberíamos hacerlo empleando, dentro del módulo al que pertecen las sentencias anteriores, distintas sentencias de asignación aritmética ya que con el COMMON con etiquetas no puede hacerse uso de la sentencia DATA que serviría para ese cometido.

Empleando la nueva sentencia BLOCK DATA, definiríamos el subprograma siguiente, con el que lograríamos el objetivo anterior.

```
BLOCK DATA INICI
DIMENSION A(10), B(5)
COMMON /BL1/ A, I, J / BL2 / B
DATA A/10*0.0/, B/5*0.0/
END
```

La forma general de esta nueva sentencia es:

siendo *nombre* el nombre adjudicado al subprograma BLOCK DATA. Este nombre puede faltar, es decir, es opcional, pero cuando existe no puede repetirse en un mismo programa ejecutable (programa principal y subprogramas).

Algunas observaciones a tener en cuenta al emplear esta sentencia, son las siguientes:

– El nombre opcional para denominar el BLOCK DATA, es un nombre global y por consiguiente no debe coincidir con el de una función externa, con el de un programa principal, con el de un COMMON con etiqueta o con el de otro BLOCK DATA del mismo programa ejecutable.

- El subprograma BLOCK DATA únicamente puede emplearse en conjunción con un COMMON con etiquetas.
- En un programa ejecutable no puede existir más que un sólo BLOCK DATA sin nombre.

Un nuevo ejemplo del uso de este subprograma sería:

```
BLOCK DATA
COMMON / BL1 / A, B, C, J, K
LOGICAL A, B
COMPLEX C
DATA A, B, C. J, K / · TRUE · , · FALSE ·, (2., –3), 5, 7 /
END
```

Por último, también sería válido escribir

```
BLOCK DATA
COMMON / BL1 / A, B
COMMON /BL2 / C, D
CHARACTER * 4 C, D
DATA A, D / 3.5, 'CASA'/
```

en el que puede observarse que las variables tipo *carácter* están incluidas en un COMMON con etiqueta diferente al que contiene otros valores reales. Además, no es necesario inicializar todas las variables contenidas en el COMMON. En nuestro ejemplo únicamente han sido inicializadas la A y la D.

EJERCICIOS RESUELTOS

8.1. Dada la llamada a la función externa BETA, decir qué tipo de llamada se produciría para cada uno de los argumentos verdaderos que la forman.

```
BETA (A, B + C, (D), E, 2.0)
```

Respuesta

Para A y E, una llamada por dirección
Para B + C, (D) y 2.0, una llamada por valor

8.2. Idem que el ejercicio anterior para la subrutina ALFA.

```
CALL ALFA (3.0, U, 2.*V, A(3), C)
```

Respuesta

Para U, A(3) y C, una llamada por dirección
Para 3.0 y 2.*V, una llamada por valor

8.3. Encontrar los errores, si existen, en cada una de las sentencias siguientes:

a) SUBROUTINE AAA (A, B(3), B(1), C)
b) SUBROUTINE BBB (A, K, B)
   DIMENSION A(J, K), B(L)
c) SUBROUTINE (2.0, A, B)

Respuesta

a) Los argumentos ficticios no pueden ser elementos de un conjunto.
b) Los subíndices J y L no están definidos como argumentos ficticios.
c) Falta el nombre de la subrutina y además un argumento ficticio no puede ser una constante.

8.4. Dado el módulo subrutina siguiente, decir en qué variable o variables se devuelven valores al programa principal:

```
SUBROUTINE AAA (A, B, C)
A = 2. * A
C = A + B
END
```

Respuesta

En las variables A y C por ser argumentos ficticios y al mismo tiempo primeros miembros de sentencias de asignación aritmética.

8.5. Dada la sentencia cabecera de una subrutina.

```
SUBROUTINE ALFA (A, B, I, TOTAL)
```

y la llamada a la misma.

```
CALL ALFA (X + Y, C(5), 5 + J, SUMA)
```

¿Cuál es la correspondencia entre argumentos verdaderos y ficticios? ¿Es correcta?

Respuesta

```
A     → X + Y
B     → C(5)
I     → 5 + J
TOTAL → SUMA
```

Sí, porque se corresponden en número, orden y tipo.

8.6. Dados los siguientes programas principal y subprograma.

```
DIMENSION A(2:10)          SUBROUTINE Z(B)
CALL Z(A)                  DIMENSION B(1:*)
```

Por qué variables están ocupadas las siguientes posiciones:

a) Ubicación 3
b) Ubicación 1
c) Ubicación 9
d) Ubicación 7

Respuesta

a) Por A(4) y B(3)
b) Por A(2) y B(1)
c) Por A(10) y B(9)
d) Por A(8) y B(7)

8.7. Dados los programa principal y subprograma siguientes:

```
CHARACTER A(9) * 9          SUBROUTINE ZETA (B)
CALL ZETA (A)               CHARACTER B(2: *) * (*)
```

Contestar a las preguntas siguientes:

a) Cuántos elementos tendrá el conjunto B
b) Cuál es el subíndice más alto de B
c) Cuántos caracteres tiene cada elemento de B
d) Qué elementos de A y B ocupan la ubicación cinco

Respuesta

a) Nueve
b) El 10
c) Nueve
d) A(5) y B(6)

8.8. Dados los programa principal y subprograma siguientes:

```
DIMENSION A(20)          SUBROUTINE B (C, I)
CALL B (A, 5)            DIMENSION C (I)
```

¿Qué elementos de A y de C ocuparán las ubicaciones tercera y sexta?

Respuesta

A(3) y C(3) en la ubicación tercera
Unicamente A(6) en la ubicación sexta

8.9. Dado el subprograma

```
SUBROUTINE BETA (A, B, I, J)
   .
   .
   .
ENTRY POINT1 (X, Z)
   .
   .
   .
ENTRY POINT2 (L, M)
   .
   .
```

Escribir sentencias CALL apropiadas para acceder a la subrutina desde su comienzo, en POINT1 y en POINT2.

Respuesta

```
CALL BETA (C, D, II, 3)
CALL POINT1 (2.*U, SQRT(3.0))
CALL POINT2 (JOTA, KA)
```

8.10. Dadas las sentencias

```
DIMENSION A(4), B(1,7)
EQUIVALENCE (A(3), B(1,5))
```

decir qué variables ocuparían las ubicaciones siguientes:

a) La 1
b) La 5
c) La 7
d) La 3

Respuesta

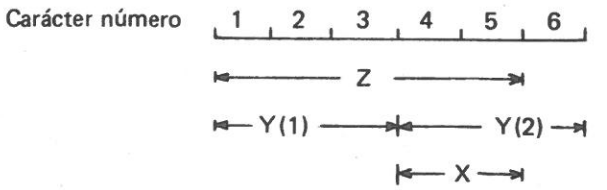
a) B(1,1)
b) A(3) y B(1,5)
c) B(1,7)
d) A(1) y B(1,3)

8.11. Dadas las sentencias siguientes:

```
CHARACTER X * 2, Y(2) * 3, Z * 5
EQUIVALENCE (Z, Y(1)), (Y(2), X)
```

¿Cuál sería el esquema de equivalencias entre elementos carácter?

Respuesta



8.12. Dadas las sentencias

```
DIMENSION A(4), C(2), D(6)
COMMON A, X, C
EQUIVALENCE (A(3), D(2))
```

Decir qué variables contienen las ubicaciones siguientes del área común: la uno, la tres, la cinco, la siete. ¿Se ha ampliado el COMMON?

Respuesta

A(1)
A(3) y D(2)
X y D(4)
C(2) y D(6)

No

8.13. Dados los dos subprogramas que siguen

```
SUBROUTINE ALFA        SUBROUTINE BETA
COMMON A(10, 10)       COMMON B(100)
```

Qué variables contendrían las siguientes ubicaciones del COMMON

a) La uno
b) La cien
c) La cincuenta
d) La ochenta

Respuesta

a) A(1,1) y B (1)
b) A(10,10) y B(100)
c) A(10,5) y B(50)
d) A(10,8) y B(80)

8.14. Escribir en una sola sentencia las parejas siguientes:

a)
```
COMMON U1, U2, U3
COMMON /BETA / L1, L2 / GAMA / A, B, C, D
```

b)
```
COMMON U1, U2
COMMON /A/ L, M, N / B / O, P, Q
```

Respuesta

a) COMMON / BETA / L1, L2 / GAMA / A, B, C, D // U1, U2, U3
b) COMMON / A / L, M, N / B / O, P, Q // U1, U2

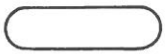
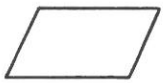
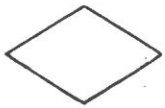
APENDICE 1

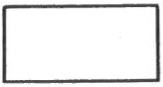
Organigramas – Símbolos empleados en el texto

A lo largo de este libro se han empleado determinados esquemas denominados organigramas u ordinogramas. Con ellos se representa el flujo lógico que sigue un procedimiento o algoritmo para llegar a la solución de un problema.

Los organigramas se componen de distintos recuadros o cajas, cada uno de los cuales con una significación distinta. Los utilizados en este texto están de acuerdo con los standards marcados por ECMA – European Computer Manufacturers Association.

A continuación se indican estos recuerdos y su significado respectivo.

SIMBOLO	SIGNIFICADO
	*Terminal:* Punto de comienzo, START, o final, STOP, en un programa.
	*E/S:* Entrada/Salida general, es decir, representa la información que llega al sistema para ser procesada, INPUT, o bien la producida como resultado, OUTPUT, por el sistema.
	*Decisión:* Indica una operación que determina varios caminos alternativos a seguir.

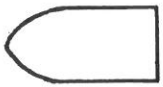
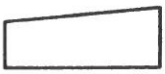
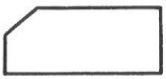


Proceso: Cualquier proceso como: operaciones aritméticas, inicialización de una variable con un cierto valor, mover el contenido de un campo A a otro B, establecer contadores dentro de un proceso.

Conector: Indica una conexión desde un punto del organigrama o otro.

Línea de flujo: Une entre sí los distintos símbolos del organigrama, indicando la dirección del flujo o de la lógica.

Es muy frecuente también la utilización de otros símbolos (símbolos del sistema) para concretar el tipo de entrada o de salida que se emplea. Los más frecuentes son los siguientes:

SIMBOLO	SIGNIFICADO
	Salida de un resultado por impresora.
	Utilización de la pantalla para la entrada/salida de la información.
	Entrada manual a través del teclado del terminal pantalla-teclado del usuario.
	Utilización de la ficha para la entrada/salida de la información.

Empleo de la cinta magnética como memoria auxiliar.

Empleo del disco magnético como memoria auxiliar.

APENDICE 2

Resumen de las sentencias FORTRAN 77

La relación que sigue contiene las sentencias disponibles en el lenguaje FORTRAN 77 completo.

SENTENCIA	COMENTARIOS	REFERENCIA
ASSIGN TO	Asignación de etiquetas a variables	Cap. 6
BACKSPACE	Posicionamiento de un fichero: retroceder	Cap. 3
BLOCK DATA	Subprograma especial	Cap. 8
CALL	Referencia a una subrutina	Cap. 8
CHARACTER	Definición tipo *carácter*	Cap. 1
CLOSE	Cerrar un fichero	Cap. 3
COMMON	Definición de área común	Cap. 8
COMPLEX	Definición tipo *complejo*	Cap. 6
CONTINUE	Sentencia sin efecto	Cap. 2
DATA	Asignación de valores iniciales a las variables	Cap. 6
DIMENSION	Definición de conjuntos	Cap. 2
DO	Ejecución y control de iteraciones	Cap. 2

DOUBLE PRECISION	Definición tipo *doble precisión*	Cap. 1
ELSE	Bloque IF	Cap. 5
ELSE IF THEN	Bloque IF	Cap. 5
END	Fin físico de un programa principal o subprograma	Cap. 2
END IF	Bloque IF: sentencia final	Cap. 5
END FILE	Fin de fichero	Cap. 3
ENTRY	Entradas múltiples en un subprograma	Cap. 8
EQUIVALENCE	Definición de variables equivalentes	Cap. 8
EXTERNAL	Definición de variables como procedimientos externos	Cap. 7
FORMAT	Definición del formato de los datos	Cap. 4
FUNCTION	Sentencia comienzo de una función	Cap. 7
GO TO	Bifurcación incondicional	Cap. 2
GO TO calculado	Bifurcación condicional	Cap. 2
GO TO asignado	Bifuración condicionada dependiente de la sentencia ASSIGN TO asociada	Cap. 6
IF aritmético	Bifuración condicionada	Cap. 2
IF lógico	Bifuración condicionada	Cap. 5
IF THEN	Bloque IF: sentencia inicial	Cap. 5
IMPLICIT	Definición del tipo de las variables	Cap. 6
INQUIRE	Consulta sobre un fichero	Cap. 3
INTEGER	Definición tipo *entero*	Cap. 6
INTRINSIC	Definición de argumentos como funciones intrínsecas	Cap. 7
LOGICAL	Definición tipo *lógico*	Cap. 5

OPEN	Abrir un fichero	Cap. 3
PARAMETER	Definición de constantes	Cap. 6
PAUSE	Parada temporal	Cap. 2
PRINT	Transferencia de datos en salidas	Cap. 3
PROGRAM	Primera sentencia de un programa. Su uso es opcional	Cap. 2
READ	Transferencia de datos en entradas	Cap. 3
REAL	Definición tipo *real*	Cap. 6
RETURN	Devolver control desde un subprograma a un programa principal	Cap. 7
REWIND	Posicionamiento de un fichero: rebobinar	Cap. 3
SAVE	Guardar valores de variables entre procedimientos	Cap. 8
STOP	Parada definitiva. Fin lógico	Cap. 2
SUBROUTINE	Sentencia comienzo de un subrutina	Cap. 3
WRITE	Transferencia de datos en salidas	Cap. 3
v = e	Asignar *e* a la variable *v*:	
	Asignación *aritmética*	Cap. 2
	Asignación *lógica*	Cap. 5
	Asignación *carácter*	Cap. 6

APENDICE 3

Ficheros externos e internos

En el texto se ha definido el concepto de fichero como una secuencia de registros, es decir, una sucesión de valores numéricos o caracteres sin que exista necesariamente una correspondencia entre dichos valores y una represención física determinada.

Una vez más es necesario señalar aquí que en FORTRAN 77, todos los registros se sobreentiende que son lógicos, en oposición al concepto de registro físico del que también se ha hablado en el texto.

Nos interesa ahora señalar que existen dos clases de ficheros en FORTRAN 77: *externos* e *internos*. Veamos las características de unos y otros.

Un *fichero externo* es aquel cuyos datos se encuentran almacenados en medios externos, como por ejemplo, en un bloque de fichas, en una página de impresora, en un terminal pantalla-teclado, en discos o cintas magnéticos.

Otra característica es que los valores de un fichero de esta clase se encuentran separados (son externos a) del programa, que residirá en la memoria principal. Por ello, y como ya se ha descrito anteriormente, son necesarias instrucciones especiales para establecer la conexión o línea de comunicación adecuada entre el programa y su fichero externo correspondiente. De ahí las instrucciones **OPEN**, **READ** y **WRITE**.

El proceso de un fichero externo incluye dos operaciones: la transferencia desde el soporte externo a la memoria principal y el tratamiento posterior, de un solo registro lógico del fichero, cada vez. Veamos como ocurren estas dos etapas.

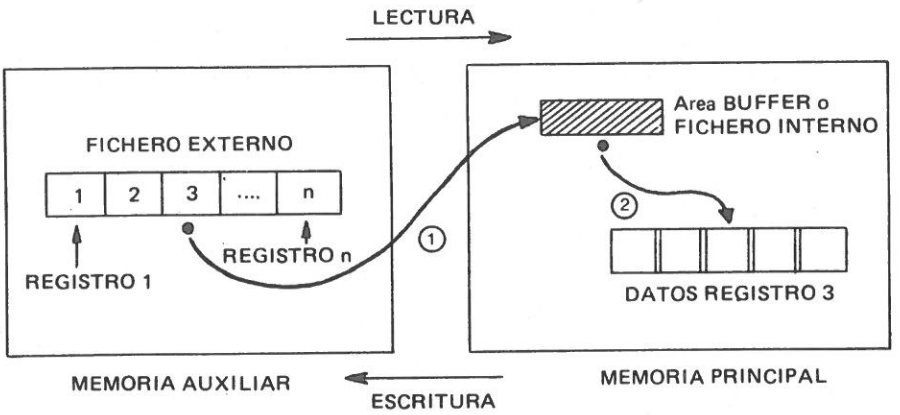
Cuando se utiliza la sentencia **READ** para leer datos, dicha sentencia da lugar a que se acceda al siguiente registro disponible del fichero contenido en la unidad de almacenamiento que se haya especificado. De igual forma, cada **WRITE** o cada **PRINT**, proporcionarán la escritura o impresión de un solo registro sobre un determinado medio. Recuérdese que el total de datos, variables, que se toman del registro de entrada (**READ**) o que se generan en la salida, (**WRITE** o **PRINT**) viene determinado, respectivamente, por la lista de E/S. Pero además, durante la transfe-

rencia, tanto en un sentido, lectura, como en otro, escritura o impresión, tiene lugar la apropiada conversión de los datos desde una forma legible para el usuario (decimal o *carácter*) a otra legible únicamente para el ordenador (binario), la cual se realiza mediante el empleo de la sentencia FORMAT.

Por último, cuando un fichero externo no contiene información, es decir, no posee registros, se dice que es un *fichero vacío.*

En algunos compiladores correspondientes al FORTRAN II y al FORTRAN 66, ya existía de alguna manera el concepto equivalente al que ahora vamos a describir sobre ficheros internos. Un *fichero interno* es un caso particular de fichero que se encuentra incorporado al FORTRAN 77 y que permite el movimiento de la información entre dos áreas de la memoria principal.

Físicamente, un fichero interno es una determinada zona de memoria principal definida por el programador como variable, conjunto, elemento/s de un conjunto siempre de tipo *carácter* o como una serie de caracteres.



Funcionamiento de un fichero interno

Siguiendo la propia definición contenida en los standars, un fichero interno es un medio de transferir y de convertir datos desde un almacenamiento interno a otro también interno.

Observando la figura adjunta, veamos como se suceden las distintas etapas empleando un área *buffer* o fichero interno, técnica que está disponible para el usuario del FORTRAN 77. Realmente, cualquier operación de E/S, utiliza esta técnica que por otra parte queda transparente o desconocida para el programador. Ahora bien; FORTRAN 77 le da la posibilidad de especificar y controlar las operaciones que vamos a describir a continuación.

Una operación de lectura, por ejemplo, desde un fichero externo, contenido en una memoria auxiliar, requiere dos etapas:

① Transferencia de los datos, registro a registro, al *buffer*, el cual puede contener uno o más registros.

② Transferencia y conversión de los datos desde el *buffer* al nuevo registro para iniciar su proceso.

Lo mismo, pero en sentido contrario, sucedería con la escritura.

La operación que hemos marcado con ① es una operación relativamente lenta (lectura de una cinta magnética, un disco, fichas) que precisa ninguna o muy poca conversión en cuanto a su representación en uno y otro medio.

La operación ② se realiza de una forma muy rápida, como toda operación dentro de la memoria principal, a la vez que los datos se convierten a una forma interna apropiada (binario). En la figura se ha representado la lectura de los datos correspondientes a un único registro.

Pues bien; cuando el programador emplea la instrucción READ que conocemos hasta aquí, lo que realmente ocurre es que se realizan las dos etapas ① y ② que acaban de describirse, de forma que los datos que estaban contenidos en el registro leído (lista de variables) quedan listos para su proceso con otros.

Pero como ya anticipamos, FORTRAN 77 dispone de instrucciones con las que el programador puede realizar, cuando convenga, la operación ② en uno y otro sentido, lectura o escritura.

Dichas operaciones se indican mediante instrucciones de E/S, READ o WRITE secuenciales acompañadas de formato FORMAT pero especificando fichero interno en lugar de externo. Este método excluye el empleo de instrucciones tales como OPEN, INQUIRE o CLOSE. No es necesario tampoco el empleo de sentencias para ficheros secuenciales como REWIND, BACKSPACE o ENDFILE puesto que un fichero interno o *buffer* siempre se encuentra como posicionado en su origen.

En el antiguo FORTRAN II estas operaciones se lograban con el empleo de la sentencia READ TAPE número *cero*, en el FORTRAN 66 con la READ sobre la unidad *cero* y en otros compiladores con el uso de sentencias específicas como ENCODE, DECODE.

Veamos ahora como se realizarían estas dos operaciones con el empleo de sentencias FORTRAN 77.

a) Definición del fichero interno o *buffer* que denominaremos FICHEIN. Si este fichero tuviera 80 caracteres, quedaría definido así:

```
CHARACTER FICHEIN*80
```

b) Lectura u operación ① desde una cinta magnética, por ejemplo, leyendo registros de longitud 80 caracteres y con formato. Para ello emplearíamos la sentencia

```
READ (5, '(10F8.2)', REC=80) FICHEIN
```

que significa: leer desde la unidad 5 (cinta) un registro de 80 caracteres formado por 10 campos de tipo real, cada uno de 8 caracteres. Estos datos quedarán en memoria almacenados en el campo FICHEIN: fichero interno. En este caso FICHEIN se emplea como una variable normal.

c) Lectura u operación②desde el fichero interno FICHEIN a los elementos especificados en la lista de variables A, B, etc.

```
READ (FICHEIN, 100) A, B, C, D, E, F, G, H, X, Y
```

En este caso FICHEIN indica un fichero interno y 100 es el número de la correspondiente sentencia de formato.

d) El efecto de estas dos últimas sentencias puede conseguirse también, como ya sabemos, empleando la única READ así:

```
READ (5, 100, REC=80) A, B, C, D, E, F, G, H, X, Y
```

Las sentencias equivalentes para escribir resultados, serían:

```
CHARACTER FICHEIN*80
WRITE (FICHEIN, 100) A, B, C, D, E, F, G, H, X, Y
WRITE (5, '(10F8.2)', REC=80) FICHEIN
```

que de una forma más simplificada podrían escribirse de este modo:

```
CHARACTER FICHEIN*80
WRITE (5, 100, REC=80) A, B, C, D, E, F, G, H, X, Y
```

Veamos ahora algunas aplicaciones prácticas sobre el uso de ficheros internos.

Ejemplo 1: En el proceso de un fichero es muy frecuente la lectura o escritura de varios registros a un mismo tiempo con el fin de acelerar ambas operaciones ya que cada una de ellas consume una cantidad apreciable de tiempo. Es lo que se denomina lectura/escritura por bloques de registros.

Para el bloqueo de registros conviene el uso del fichero interno.

Supongamos un conjunto de 10 registros contenidos en una cinta magnética (unidad 5), cada uno de ellos formado por dos campos reales de diez caracteres y otros dos enteros de cinco. Estos registros se leen dentro del fichero interno FICHEIN, se procesan luego uno a uno, se restituyen a continuación al fichero interno y por fin de devuelven a la cinta megnética externa como un nuevo bloque actualizado. La codificación podría ser:

```
      INTEGER REG
      CHARACTER*300 FICHEIN (10)
      READ (5, '10 (F10.3, F10.3, I5, I5) ' ) FICHEIN
        DO 1 REG = 1,10
        READ (FICHEIN (REG : REG), 100) A, B, I, J

          .
          .
          .   } Proceso del registro (cálculo, actualización, etc.)
          .
          .
  1     WRITE (FICHEIN (REG : REG) , 100 ) A, B, I, J
      BACKSPACE 5
      WRITE (5, '10 (F10.3, F10.3, I5, I5) ' ) FICHEIN
100   FORMAT .....
```

Ejemplo 2: Se trata ahora de leer un fichero que contiene diversos registros pero el formato de cada uno de ellos no es conocido a priori sino que depende de una clave que cada registro contiene como último campo del mismo.

Supongamos, por ejemplo, que cada registro está formado por 50 dígitos que pueden corresponde a 50 campos de *un* dígito, a 25 de *dos* dígitos o a 10 de *cinco*, siempre campos enteros. La información sobre qué clase de datos van a procesarse, viene dada mediante un dígito clave 1, 2 ó 5, respectivamente, contenido al final del registro, como ya hemos dicho. Si los registros estuvieran almacenados en una cinta magnética 5, la codificación podría ser, indicando FICHEIN el fichero interno, la siguiente:

```
INTEGER ENT (50), CLAVE
CHARACTER FICHEIN*50, FORMA (3)*6
DATA FORMA / ' (50I1)', '(25I2)', '(10I5)' /
READ (5, '(A50, I1)' FICHEIN, CLAVE
READ (FICHEIN, FORMA (CLAVE)) (ENT (I) = 1,50 / CLAVE)
```

Las variables indican lo siguiente:

ENT, conjunto de cincuenta elementos que recibirá todos los valores numéricos.

FORMA, conjunto de tres elementos, cada uno de los cuales contiene un formato posible, 50I1 ó 25I2 ó 10I5.

FICHEIN, es el nombre del *buffer* o fichero interno.

Existen dos sentencias READ: la primera se encarga de leer los cincuenta dígitos y la clave desde la cinta magnética 5. La segunda transfiere esos datos desde el fichero interno a ENT, según el bucle implícito cuyo valor final 50/CLAVE es una expresión entera que dependerá, en cualquier caso, del valor contenido en la variable definida como entera CLAVE: si vale uno, el valor final será 50; si vale 2, 25 y si vale 5, dicho valor final será 10.

APENDICE 4

Estructura de las sentencias Fortran 77

Abreviaturas:

Argum1	Argumento verdadero
Argum2	Argumento ficticio
Conjts	Conjuntos
Consts	Constantes
Etiq	Etiqueta
Expr	Expresión
Expraritme	Expresión aritmética
Exprentera	Expresión entera
Exprlogica	Expresión lógica
Senten	Sentencia
Varcaract	Variable carácter
Varcomple	Variable compleja
Vardopre	Variable doble precisión
Varen	Variable de entrada
Varentera	Variable entera
Varlog	Variable lógica
Varreal	Variable real
Varsa	Variable de salida
Vars	Variables

Significado de los símbolos:

<...>	Dato suministrado por el usuario
<lista...>	Varios elementos separados por comas
[...]	Parte opcional
{...}	Serie de uno o más elementos

SENTENCIA	COMENTARIO
Ejecutables	
• <variable> = <expr>	Asignar una expresión a una variable
• ASSIGN <etiq> TO <varentera> { <senten ejecutable> } GO TO <varentera>[, (<lista etiq>)]	
• BACKSPACE <dispositivo [,lista]>	Retroceder un registro en un fichero SAM. [lista] contiene parámetros opcionales
• CALL <nombre>[(<lista argum1>)]	<nombre> indica la denominación del procedimiento
• CLOSE (<dispositivo>,<lista close>)	Desconectar un fichero de un dispositivo de E/S. <lista close> es una relación de opciones
• [<etiq>] CONTINUE	
DO <etiq>[,]<variación indice> { <senten ejecutable> } <etiq> CONTINUE	Control de bucles. Es aconsejable finalizar la DO con la sentencia CONTINUE
• ELSE	Ver bloques IF lógicos
• ELSE IF (<exprlogica>) THEN	Ver Bloque IF lógico compuesto
• END	Final físico de programa
• END FILE <dispositivo>[,<lista>]	Grabar registro fin de fichero. <lista> contiene una relación de opciones
• GO TO <etiq>	Bifurcación incondicional
• GO TO <lista etiq>[,]<exprentera>	Bifurcación calculada

SENTENCIA	COMENTARIO
• IF (<expraritme>)<etiq>,<etiq>,<etiq>	IF aritmético
• IF (<exprlogica>)<senten ejecutable> <senten ejecutable>	IF lógico
• IF (<exprlogica>) THEN {<senten ejecutable>} [ELSE {<senten ejecutable>}]	Bloque IF lógico simple

• IF (<exprlogica>) THEN
{<senten ejecutable>}
{ELSE IF (<exprlogica>) THEN
{<senten ejecutable>}}
ELSE
END IF
	Bloque IF lógico compuesto

• INQUIRE (<fichero>,<lista>)	Obtener información de un fichero. <lista> contiene una relación de opciones
• INQUIRE (<unidad>,<lista>)	Obtener información de una unidad de E/S. <lista> contiene una relación de opciones
• OPEN (<dispositivo>,<lista open>)	Conectar un fichero a una unidad de E/S. <lista open> es una relación de opciones
• PAUSE [<constante>]	
• PRINT * [,<lista varsa>]	Impresión de línea en blanco o de expresiones
• PRINT <etiq>,<lista expr salida>	Sentencia básica de Salida. <etiq> se relaciona con una sentencia de formato. <lista expr salida> indica tanto variables como expresiones, precedidas o no de constantes carácter.
• PRINT '(<lista format>)',<lista varsa>	Sentencia básica de Salida

SENTENCIA	COMENTARIO
• READ *,<lista varen>	Sentencia básica de Entrada
• READ <etiq>,<lista varen>	Sentencia básica de Entrada, <etiq> corresponde a una sentencia de formato
• READ '(<lista format>)',<lista varen>	Sentencia básica de Entrada
• READ (<lista control>) <lista varen>	Sentencia general de Entrada. <lista control> puede contener: formato, identificación del fichero, recuperación por error, etc.
• RETURN [<expresión entera>]	
• REWIND <dispositivo [,lista]>	Posicionar en su primer registro un fichero SAM. [lista] contiene una relación de opciones
• STOP [<constante>]	
• WRITE (<lista control>) <lista varsa>	Sentencia general de Salida. <lista control> puede contener: formato, identificación del fichero, recuperación por error, etc.
No Ejecutables	
• BLOCK DATA [<nombre bloque>]	Sentencia primera de un procedimiento utilizada en conjunción con las COMMON con etiquetas y DATA
• COMMON [/<etiq>/] <lista vars>	Definición de una zona común a varios módulos
• COMPLEX <lista varcomple>	
• CHARACTER [*<long>]<lista varcaract>	

SENTENCIA	COMENTARIO
• DATA <lista vars>/<lista consts>/ [[,<lista vars>/<lista consts>]]	Asignación de valores constantes iniciales a variables
• DIMENSION <lista declaración conjts>	
• DOUBLE PRECISION <lista vardopre>	
• ENTRY <nombre entrada>[<lista argum2>]	Punto de entrada variable de un procedimiento SUBROUTINE o FUNCTION
• EQUIVALENCE {(<lista vars>)}	
• EXTERNAL <lista procedimientos>	
• <etiq> FORMAT (<lista formatos>)	
• FUNCION (<lista variables>)=<expr>	Estructura para definir una función de sentencia
• [<tipo>] FUNCTION <nombre> [(<lista argum2>)]	<nombre> indica la denominación del procedimiento
• IMPLICIT {<tipo(carácter alfabético)>}	
• INTEGER <lista varentera>	
• INTRINSIC <lista funciones intrínsecas>	
• LOGICAL <lista varlog>	
• PARAMETER (<lista variable=expr>)	
• PROGRAM <nombre programa>	Primera sentencia de un programa. Es opcional
• REAL <lista varreal>	
• SAVE <lista variables locales>	Conserva las variables locales de los distintos procedimientos entre las diversas llamadas
• SUBROUTINE <nombre> [<lista argum2>]	<nombre> indica la denominación del procedimiento

APENDICE 5

Algoritmos programados

Un algoritmo está constituido por un conjunto de reglas encaminadas a la solución de un problema a través de una serie definida y finita de etapas. Los elementos que forman un algoritmo son tres: los **datos** de base, el **procedimiento** mediante el cual se transforman los datos de partida y los **resultados** que desean obtenerse.

Para llegar a la solución de un problema mediante la utilización de un algoritmo y en consonancia con la definición anterior, han de recorrerse tres etapas: la **preparación del tratamiento** en la cual se examinan los datos, su coherencia y su validez; el **tratamiento** propiamente dicho donde el procedimiento de que hemos hablado se subdivide en las etapas necesarias; la **obtención de resultados** mediante los cuales, el usuario del algoritmo, podrá conocer la solución del problema cuya solución buscaba.

A continuación se presenta una serie de ejercicios clásicos de programación resueltos mediante algoritmos expresados en el lenguaje FORTRAN 77. En cada uno de ellos, y después del código, se listan los resultados obtenidos con la ejecución.

ALGORITMO 1

Sumar tres cantidades cuyos identificadores son las tres variables reales X, Y, Z.

Las sentencias empleadas para la E/S son las básicas de que dispone el lenguaje FORTRAN.

```
      PROGRAM PR01
      PRINT *,'SUMA DE TRES CANTIDADES'
C
C     PREPARACION
C
      PRINT *,'TECLEAR TRES CANTIDADES'
      READ *,X,Y,Z
C
C     PROCEDIMIENTO
C
      SUMA=X+Y+Z
C
C     RESULTADO
C
      PRINT *,SUMA
      STOP
      END
```

Algoritmo 1

```
SUMA DE TRES CANTIDADES
TECLEAR TRES CANTIDADES
3
4
-6

      1.00000000
```

```
SUMA DE TRES CANTIDADES
TECLEAR TRES CANTIDADES
3.0
4.0
-6.0

      1.00000000
```

ALGORITMO 2

Conocido el radio de un círculo, calcular su área.

RADIO es el identificador empleado para denominar el objeto "radio" del círculo y sus valores son introducidos en la etapa de preparación del algoritmo. SUPER será el identificador correspondiente al valor de la superficie.

Es importante elegir los identificadores de forma que recuerden el objeto o concepto que representan.

```
      PROGRAM PR02
      PRINT *,'SUPERFICIE DE UN CIRCULO'
C
C     PREPARACION
C
      PRINT *,'RADIO= '
      READ *,RADIO
C
C     PROCEDIMIENTO
C
      PI=3.1416
      SUPER=PI*RADIO**2
C
C     RESULTADO
C
      PRINT '(A,F10.3)',' SUPERFICIE= ',SUPER
      STOP
      END
```

Algoritmo 2

```
SUPERFICIE DE UN CIRCULO
RADIO= 2.34567893

SUPERFICIE=      17.286


SUPERFICIE DE UN CIRCULO
RADIO= 1.5

SUPERFICIE=       7.069
```

ALGORITMO 3

Calcular la suma de los elementos de una progresión aritmética así como el valor del último término, sabiendo que todos ellos son valores enteros.

Los datos del problema son: el primer término I1, la razón K y el número total de elementos N.

```
      PROGRAM PR03
      PRINT *,'PROGRESIONES ARITMETICAS'
      PRINT *,'FACILITAR DATOS ENTEROS'
C
C     PREPARACION
C
      PRINT *,'ELEMENTO 1= '
      READ *,I1
      PRINT *,'RAZON= '
      READ *,K
      PRINT *,'NUMERO DE TERMINOS= '
      READ *,N
C
C     PROCEDIMIENTO
C
      IN=I1+(N-1)*K
      J=((I1+IN)*N)/2
C
C     RESULTADO
C
PRINT *,'ULTIMO TERMINO= ',IN
PRINT *,'SUMA DE LOS TERMINOS= ',J
STOP
END
```

Algoritmo 3

```
PROGRESIONES ARITMETICAS
FACILITAR DATOS ENTEROS
ELEMENTO 1= 2

RAZON= 3

NUMERO DE TERMINOS= 20

ULTIMO TERMINO=          59
SUMA DE LOS TERMINOS=          610

PROGRESIONES ARITMETICAS
FACILITAR DATOS ENTEROS
ELEMENTO 1= 420

RAZON= -3

NUMERO DE TERMINOS= 100

ULTIMO TERMINO=         123
SUMA DE LOS TERMINOS=       27150
```

ALGORITMO 4

Dados dos valores cualesquiera, cuyos identificadores son X e Y, en un orden no determinado, escribir un procedimiento para cambiar dicho orden.

Mediante el identificador AUX se designa una variable auxiliar donde se almacena el primero de los valores X antes de pasar el segundo Y a la posición ocupada por el primero.

```
      PROGRAM PR04
      PRINT *,'PERMUTACION DE DOS VALORES'
C
C     PREPARACION
C
      PRINT *,'TECLEAR DOS VALORES '
      READ '(2F6.0)',X,Y
      PRINT '(2(A,F6.2))',' X= ',X,'   Y= ',Y
C
C     PROCEDIMIENTO
C
      AUX=X
      X=Y
      Y=AUX
C
C     RESULTADO
C
      PRINT '(2(A,F6.2))',' X= ',X,'   Y= ',Y
      STOP
      END
```

Algoritmo 4

```
PERMUTACION DE DOS VALORES
TECLEAR DOS VALORES 2.3400 56.890

X=   2.34   Y=  56.89
X=  56.89   Y=   2.34


PERMUTACION DE DOS VALORES
TECLEAR DOS VALORES 2.3000 5.8000

X=   2.30   Y=   5.80
X=   5.80   Y=   2.30
```

ALGORITMO 5

Dados tres valores X, Y, Z, permutarlos circularmente.

Se trata de producir un movimiento entre los elementos tal que si los datos de partida están dispuestos en el orden

X Y Z

el orden definitivo sea

Y Z X

```
      PROGRAM PR05
      INTEGER X,Y,Z,AUX
      PRINT *,'PERMUTACIÓN CIRCULAR DE TRES ELEMENTOS'
C
C     PREPARACIÓN
C
      PRINT *,'TECLEAR TRES VALORES ENTEROS '
      READ '(3I4)',X,Y,Z
      PRINT '(3(A,I4))',' X= ',X,'    Y= ',Y,'    Z= ',Z
C
C     ALGORITMO
C
      AUX=X
      X=Y
      Y=Z
      Z=AUX
C
C     RESULTADO
C
      PRINT '(3(A,I4))',' X= ',X,'    Y= ',Y,'    Z= ',Z
      STOP
      END
```

Algoritmo 5

```
PERMUTACIÓN CIRCULAR DE TRES ELEMENTOS
TECLEAR TRES VALORES ENTEROS    0   4  19

X=    0    Y=    4    Z=   19
X=    4    Y=   19    Z=    0


PERMUTACIÓN CIRCULAR DE TRES ELEMENTOS
TECLEAR TRES VALORES ENTEROS    3   5  -7

X=    3    Y=    5    Z=   -7
X=    5    Y=   -7    Z=    3


PERMUTACIÓN CIRCULAR DE TRES ELEMENTOS
TECLEAR TRES VALORES ENTEROS    1  78   0

X=    1    Y=   78    Z=    0
X=   78    Y=    0    Z=    1
```

ALGORITMO 6

Dadas dos cantidades cualesquiera, encontrar la mayor de ellas.

Los identificadores para las cantidades serán X e Y; la mayor de ellas se almacenará en SUPER.

```
      PROGRAM PR06
      PRINT *,'ESCRIBIR EL MAYOR DE DOS ELEMENTOS'
C
C     PREPARACION
C
      PRINT *,'TECLEAR DOS VALORES '
      READ '(F6.2,1X,F6.2)',X,Y
C
C     PROCEDIMIENTO
C
      IF(X.GT.Y)THEN
                 SUPER=X
               ELSE
                SUPER=Y
      END IF
C
C     RESULTADO
C
      PRINT '(A,F6.2)',' MAYOR=',SUPER
      STOP
      END
```

Algoritmo 6

```
ESCRIBIR EL MAYOR DE DOS ELEMENTOS
TECLEAR DOS VALORES   2.00   3.00

MAYOR=  3.00

ESCRIBIR EL MAYOR DE DOS ELEMENTOS
TECLEAR DOS VALORES  -7.00  -8.00

MAYOR= -7.00
```

ALGORITMO 7

Haciendo uso del algoritmo anterior, determinar el mayor de tres valores diferentes X, Y, Z.

El algoritmo, en la etapa de preparación, comprueba que efectivamente no existan dos valores iguales.

```
      PROGRAM PR07
      PRINT *,'ESCRIBIR EL MAYOR DE TRES VALORES'
C
C     PREPARACION
C
1     PRINT *,'ESCRIBIR TRES VALORES DIFERENTES '
      READ '(3(F6.2,1X))',X,Y,Z
      IF (X.EQ.Y.OR.Y.EQ.Z.OR.X.EQ.Z) THEN
         PRINT *,'LOS DATOS SON ERRÓNEOS'
         PRINT *,'TECLEAR NUEVOS DATOS'
         GO TO 1
      END IF
C
C     PROCEDIMIENTO
C
      IF (X.GT.Y) THEN
         SUPER=X
           ELSE
         SUPER=Y
      END IF
      IF (Z.GE.SUPER) THEN
         SUPER=Z
           ELSE
      END IF
C
C     RESULTADO
C
      PRINT '(A,F6.2)',' MAYOR= ',SUPER
      STOP
      END
```

Algoritmo 7

```
ESCRIBIR EL MAYOR DE TRES VALORES
ESCRIBIR TRES VALORES DIFERENTES   2.00   3.00   5.00

MAYOR=   5.00

TECLEAR NUEVOS DATOS
ESCRIBIR TRES VALORES DIFERENTES   2.00   2.00  -9.00

LOS DATOS SON ERRÓNEOS
TECLEAR NUEVOS DATOS
ESCRIBIR TRES VALORES DIFERENTES   2.00   3.00  -9.00

MAYOR=   3.00
```

ALGORITMO 8

Dadas dos cantidades diferentes, ordenarlas en forma creciente.
El algoritmo comprueba que ambas cantidades sean distintas.

```
      PROGRAM PR08
      PRINT *,'ORDENAR DOS CANTIDADES EN FORMA CRECIENTE'
C
C     PREPARACIÓN
C
1     PRINT *,'ESCRIBIR CANTIDADES A ORDENAR '
      READ '(2(F6.2,1X))',X,Y
C
C     TRATAMIENTO
C
      IF (X.EQ.Y) THEN
         PRINT *,'LAS DOS CANTIDADES SON IGUALES'
         GO TO 1
      END IF
      IF (X.GE.Y) THEN
         AUX=X
         X=Y
         Y=AUX
      END IF
C
C     RESULTADO
C
      PRINT '(2(A,F6.2))',' CANTIDADES X=',X,' Y=',Y,' ESTAN ORDENADAS'
      STOP
      END
```

Algoritmo 8

```
ORDENAR DOS CANTIDADES EN FORMA CRECIENTE
ESCRIBIR CANTIDADES A ORDENAR   2.76   5.76

CANTIDADES X=  2.76 Y=  5.76
ESTAN ORDENADAS

ORDENAR DOS CANTIDADES EN FORMA CRECIENTE
ESCRIBIR CANTIDADES A ORDENAR   2.56   2.56

LAS DOS CANTIDADES SON IGUALES
ESCRIBIR CANTIDADES A ORDENAR   1.00  -4.98

CANTIDADES X= -4.98 Y=  1.00
ESTAN ORDENADAS
```

ALGORITMO 9

Clasificar en orden creciente tres cantidades X, Y, Z.

Se trata de una aplicación basada en los dos algoritmos anteriores que detecta, como error, la igualdad entre las cantidades datos.

```
      PROGRAM PR09
      PRINT *,'CLASIFICAR TRES NUMEROS EN ORDEN CRECIENTE'
C
C     PREPARACION
C
1     PRINT *,'INTRODUCIR TRES VALORES DISTINTOS'
      READ '(3(F6.2,1X))',X,Y,Z
      IF (X.EQ.Y.OR.Y.EQ.Z.OR.X.EQ.Z)THEN
         PRINT *,'LOS DATOS SON ERRONEOS'
         GO TO 1
      END IF
C
C     TRATAMIENTO
C
      IF (X.GT.Y) THEN
         AUX=X
         X=Y
         Y=AUX
      END IF
      IF (X.GT.Z) THEN
         AUX=X
         X=Z
         Z=AUX
      END IF
      IF (Y.GT.Z) THEN
         AUX=Y
         Y=Z
         Z=AUX
      END IF
C
C     RESULTADO
C
      PRINT '(3(A,F6.2))',' EL ORDEN ES: X= ',X,'  Y= ',Y,'  Z= ',Z
      STOP
      END
```

Algoritmo 9

```
CLASIFICAR TRES NUMEROS EN ORDEN CRECIENTE
INTRODUCIR TRES VALORES DISTINTOS  2.00   2.00   2.00

LOS DATOS SON ERRONEOS
INTRODUCIR TRES VALORES DISTINTOS  3.00   1.00  -4.00

EL ORDEN ES: X=  -4.00  Y=   1.00  Z=   3.00
```

ALGORITMO 10

Sin efectuar cálculo aritmético alguno, averiguar el signo del producto de dos cantidades cualesquiera.

Los identificadores de las cantidades son X e Y y la respuesta del algoritmo será: POSITIVO o NEGATIVO.

```
      PROGRAM PR10
      INTEGER SIGNO
      PRINT *,'SIGNO DEL PRODUCTO DE DOS NÚMEROS'
C
C     PREPARACIÓN
C
      PRINT *,'TECLEAR DOS VALORES CUALESQUIERA '
      READ *,X,Y
C
C     TRATAMIENTO
C
      IF (X.GT.0) THEN
         IF (Y.GE.0) THEN
            SIGNO=1
            ELSE
            SIGNO=0
         END IF
         ELSE
         IF (Y.GT.0) THEN
            SIGNO=0
            ELSE
            SIGNO=1
         END IF
      END IF
C
C     RESULTADO
C
      PRINT *,'EL PRODUCTO ES: '
      IF (SIGNO.EQ.1) THEN
         PRINT *,'POSITIVO'
         ELSE
         PRINT *,'NEGATIVO'
      END IF
      STOP
      END
```

Algoritmo 10

```
SIGNO DEL PRODUCTO DE DOS NÚMEROS
TECLEAR DOS VALORES CUALESQUIERA -1.1,2

EL PRODUCTO ES:
NEGATIVO

SIGNO DEL PRODUCTO DE DOS NÚMEROS
TECLEAR DOS VALORES CUALESQUIERA -2.3,-8.0

EL PRODUCTO ES:
POSITIVO
```

ALGORITMO 11

Se dan tres cantidades X, Y, Z, que deben estar ordenadas en forma creciente. Dada otra cantidad cualquiera A encontrar los tres valores más pequeños entre los cuatro X, Y, Z y A.

```
      PROGRAM PR11
      WRITE (*,100)
C
C     PREPARACION
C
      WRITE  (*,101)
      READ (*,200) X,Y,Z
      IF (Y.GT.X.AND.Z.GT.Y) THEN
      GO TO 10
         ELSE
         WRITE (*,102)
         WRITE (*,101)
         READ (*,200) X,Y,Z
   10    WRITE (*,103)
         READ (*,201) A
      END IF
C
C     TRATAMIENTO
C
      IF (A.LT.X) THEN
         Z=Y
         Y=X
         X=A
          ELSE
        IF (A.LT.Y) THEN
           Z=Y
           Y=A
            ELSE
          IF (A.LT.Z) THEN
             Z=A
          END IF
        END IF
      END IF
C
C     RESULTADO
C
      WRITE (*,104)
      WRITE (*,105) X,Y,Z
  100 FORMAT (1X,'ESCRIBIR LOS TRES NUMEROS MENORES ENTRE CUATRO')
  101 FORMAT (1X,'DAR TRES NUMEROS EN ORDEN CRECIENTE')
  102 FORMAT (1X,'LOS NUMEROS NO ESTAN ORDENADOS')
  103 FORMAT (1X,'DAR EL NUMERO A CLASIFICAR')
  104 FORMAT (1X,'LOS TRES NUMEROS MAS PEQUEÑOS SON:')
  105 FORMAT (1X,'X=',F8.2,' Y=',F8.2,' Z=',F8.2)
  200 FORMAT (3(F8.2,1X))
  201 FORMAT (F8.2)
      STOP
      END
```

Algoritmo 11

```
ESCRIBIR LOS TRES NUMEROS MENORES ENTRE CUATRO
DAR TRES NUMEROS EN ORDEN CRECIENTE     6.00    -8.67     0.00

LOS NUMEROS NO ESTAN ORDENADOS
DAR TRES NUMEROS EN ORDEN CRECIENTE    -1.34     2.98     2.99

DAR EL NUMERO A CLASIFICAR     2.97

LOS TRES NUMEROS MAS PEQUEÑOS SON:
X=    -1.34 Y=     2.97 Z=     2.98

ESCRIBIR LOS TRES NUMEROS MENORES ENTRE CUATRO
DAR TRES NUMEROS EN ORDEN CRECIENTE     2.00     3.89     3.90

DAR EL NUMERO A CLASIFICAR    -2.89

LOS TRES NUMEROS MAS PEQUEÑOS SON:
X=    -2.89 Y=     2.00 Z=     3.89
```

ALGORITMO 12

Calcular la suma de los N primeros números enteros positivos.

Se utiliza la variable de identificador SUMA, cuyo valor inicial es cero, para ir acumulando las distintas sumas parciales.

```
      PROGRAM PR12
      INTEGER SUMA
      WRITE (*,100)
      WRITE (*,101)
C
C     PREPARACION
C
      READ (*,102) N
C
C     TRATAMIENTO
C
      SUMA=0
      DO 1 I=1,N
         SUMA=SUMA+I
    1 CONTINUE
C
C     RESULTADO
C
      WRITE (*,103) SUMA
  100 FORMAT (1X,'CALCULAR SUMA N PRIMEROS ENTEROS')
  101 FORMAT (1X,'VALOR DE N= ')
  102 FORMAT (I2)
  103 FORMAT (1X,'VALOR DE LA SUMA=',I4)
      STOP
      END
```

Algoritmo 12

```
CALCULAR SUMA N PRIMEROS ENTEROS
VALOR DE N= 45

VALOR DE LA SUMA=1035

CALCULAR SUMA N PRIMEROS ENTEROS
VALOR DE N= 4

VALOR DE LA SUMA=  10
```

ALGORITMO 13

Calcular el factorial de un número entero y positivo N igual o menor que 9.

El algoritmo puede modificarse para emplearlo en enteros superiores a 9 sin más que variar el formato de los identificadores N y FACT, correspondiendo este último a la variable donde se almacena el resultado.

```
      PROGRAM PR13
      INTEGER FACT
C
C     PREPARACION
C
      WRITE (*,100)
      READ (*,101) N
C
C     TRATAMIENTO
C
      IF (N.LE.1) THEN
         FACT=1
         GO TO 2
      END IF
      FACT=1
      DO 1 I=2,N
         FACT=FACT*I
    1 CONTINUE
C
C     RESULTADO
C
    2 WRITE (*,102) N,FACT
  100 FORMAT (1X,'VALOR DE N= '/)
  101 FORMAT (I2)
  102 FORMAT (1X,'FACTORIAL DE ',I2,' VALE ',I8)
      STOP
      END
```

Algoritmo 13

```
VALOR DE N=
00

FACTORIAL DE  0 VALE        1

VALOR DE N=
06

FACTORIAL DE  6 VALE      720
```

ALGORITMO 14

Se define una sucesión mediante la ley

$$t_n = t_{n-1} + 2t_{n-2}$$

Dados los dos primeros términos t_1 y t_2, calcular un término cualquiera de la sucesión.

El término buscado, que debe ser igual o superior a tres, estará contenido en la variable NN, mientras que los dos primeros de la sucesión están, respectivamente, en N1 y N2.

```
      PROGRAM PR14
      WRITE (*,100)
C
C     PREPARACION
C
      WRITE (*,101)
      READ (*,102) N1,N2
      WRITE (*,103)
    1 READ (*,104) NN
      IF (NN.LT.3) THEN
         GO TO 1
      END IF
C
C     TRATAMIENTO
C
      DO 2 I=3,NN
         N=N1+2*N2
         N1=N2
         N2=N
    2 CONTINUE
C
C     RESULTADO
C
      WRITE (*,105) NN,N
  100 FORMAT (1X,'CALCULO DE UN TERMINO DE UNA SUCESION')
  101 FORMAT (1X,'ESCRIBIR DOS PRIMEROS TERMINOS SUCESION ')
  102 FORMAT (2(I2,1X))
  103 FORMAT (1X,'TERMINO BUSCADO ')
  104 FORMAT (I2)
  105 FORMAT (1X,'TERMINO ',I3,1X,'VALE ',I5)
      STOP
      END
```

Algoritmo 14

```
CALCULO DE UN TERMINO DE UNA SUCESION
ESCRIBIR DOS PRIMEROS TERMINOS SUCESION 02 03
```

```
TERMINO BUSCADO 06

TERMINO    6 VALE    111

CALCULO DE UN TERMINO DE UNA SUCESION
ESCRIBIR DOS PRIMEROS TERMINOS SUCESION 12 14

TERMINO BUSCADO 04

TERMINO    4 VALE     94
```

ALGORITMO 15

Haciendo uso del algoritmo número 13, calcular las combinaciones de m elementos tomados de n en n.

Téngase en cuenta que tanto m como n no pueden ser negativos y que m debe ser igual o mayor que n.

CMN es el identificador que corresponde al número combinatorio.

```
      PROGRAM PR15
      INTEGER FACTM,FACTN,FACTMN,CMN
C
C     PREPARACION
C
    1 WRITE (*,100)
      READ (*,101) M,N
      IF (M.LT.N.OR.M.LT.0.OR.N.LT.0) THEN
         WRITE (*,103)
         GO TO 1
      END IF
      IF (M.EQ.N) THEN
         CMN=1
         GO TO 5
      END IF
      IF (N.EQ.0) THEN
         CMN=1
         GO TO 5
      END IF
C
C     TRATAMIENTO
C
      FACTM=1
      DO 2 I=2,M
         FACTM=FACTM*I
    2 CONTINUE
      FACTN=1
      DO 3 J=2,N
         FACTN=FACTN*J
    3 CONTINUE
      FACTMN=1
      DO 4 K=2,M-N
         FACTMN=FACTMN*K
    4 CONTINUE
      CMN=FACTM/(FACTN*FACTMN)
C
C     RESULTADO
C
    5 WRITE (*,102) M,N,CMN
C
  100 FORMAT (1X,'VALORES DE M Y N ')
  101 FORMAT (2(I2,1X))
  102 FORMAT (1X,' M=',I3,'  N=',I3,'  CMN=',I5)
  103 FORMAT (1X,'DATOS INCORRECTOS')
      STOP
      END
```

Algoritmo 15

```
VALORES DE M Y N  3  5

DATOS INCORRECTOS
VALORES DE M Y N -3  5

DATOS INCORRECTOS
VALORES DE M Y N  3 -5

DATOS INCORRECTOS
VALORES DE M Y N  8  8

 M=  8  N=  8  CMN=    1

VALORES DE M Y N 11  8

 M= 11  N=  8  CMN=  165
```

ALGORITMO 16

Dado un vector de seis elementos cuyo identificador es VEC, calcular la suma de sus elementos cada uno de los cuales puede ser un valor de la forma ±xx.xx, siendo x una cifra cualquiera.

```
      PROGRAM PR16
C
C     PREPARACION
C
      DIMENSION VEC(6)
      WRITE (*,100)
      READ (*,101) VEC
C
C     TRATAMIENTO
C
      SUMA=0
      DO 1 I=1,6
         SUMA=SUMA+VEC(I)
    1 CONTINUE
C
C     RESULTADO
C
      WRITE (*,102) SUMA
  100 FORMAT (1X,'TECLEAR LOS ELEMENTOS DEL VECTOR')
  101 FORMAT (6(F6.0,1X))
  102 FORMAT (1X,'SUMA=',1X,F7.2)
      STOP
      END
```

Algoritmo 16

```
TECLEAR LOS ELEMENTOS DEL VECTOR  3.45   2.66  23.67 -87.34
        23.00   2.56

SUMA=  -32.00
```

ALGORITMO 17

Encontrar el mayor de los elementos de un vector compuesto por seis términos, todos enteros, de la forma ±xx.

```
      PROGRAM PR17
C
C     PREPARACION
C
      INTEGER VEC(6)
      WRITE (*,100)
      READ (*,101) (VEC(I),I=1,6)
C
C     TRATAMIENTO
C
      MAYOR=VEC(1)
      J=1
      DO 1 I=1,6
         IF (MAYOR.LT.VEC(I+1)) THEN
            MAYOR=VEC(I+1)
            J=I+1
         END IF
    1 CONTINUE
C
C     RESULTADO
C
      WRITE (*,102) J,MAYOR
C
  100 FORMAT (1X,'INTRODUCIR ELEMENTOS VECTOR',/)
  101 FORMAT (6(I3,1X))
  102 FORMAT (1X,'EL ELEMENTO MAYOR ES EL ',I3,' Y VALE ',I3)
      STOP
      END
```

Algoritmo 17

```
INTRODUCIR ELEMENTOS VECTOR
 12 -03  05  11 -07  08

EL ELEMENTO MAYOR ES EL   1 Y VALE  12

INTRODUCIR ELEMENTOS VECTOR
 33  45  20  18 -09  23

EL ELEMENTO MAYOR ES EL   2 Y VALE  45
```

ALGORITMO 18

Calcular el producto escalar de dos vectores cuyos identificadores respectivos son V1 y V2, cada uno formado por seis elementos reales de la forma ±XX.XX.

```
      PROGRAM PR18
C
C     PREPARACION
C
      DIMENSION V1(6),V2(6)
      WRITE(*,100)
      READ (*,101) (V1(I),I=1,6)
      WRITE (*,102)
      READ (*,101) (V2(I),I=1,6)
C
C     TRATAMIENTO
C
      ESCA=0.0
      DO 1 I=1,6
         ESCA=ESCA+V1(I)*V2(I)
    1 CONTINUE
C
C     RESULTADO
C
      WRITE (*,103) ESCA
C
  100 FORMAT (1X,'DATOS DE V1 ')
  101 FORMAT (6(F6.0,1X))
  102 FORMAT (1X,'DATOS DE V2 ')
  103 FORMAT (1X,'EL PRODUCTO ESCALAR ',/,' DE V1 Y V2 VALE ',/,1X,F7.2)
      STOP
      END
```

Algoritmo 18

```
DATOS DE V1  23.00 -45.08  03.78  05.00  10.67 -12.56

DATOS DE V2  12.34 -23.18  00.00  45.64  32.12 -10.05

EL PRODUCTO ESCALAR
DE V1 Y V2 VALE
2025.92

DATOS DE V1  00.00  00.00  00.00  00.00  01.11  02.22

DATOS DE V2  11.00  22.00  33.00 -00.11 -08.80  00.22

EL PRODUCTO ESCALAR
DE V1 Y V2 VALE
  -9.28
```

ALGORITMO 19

Calcular el producto vectorial de dos vectores V1 y V2 compuesto cada uno por tres elementos enteros de la forma ±xx. Almacenar las tres coordenadas del vector resultante en V3.

```
      PROGRAM PR19
C
C     PREPARACION
C
      INTEGER V1(3),V2(3),V3(3)
      WRITE (*,100)
      READ (*,101) (V1(I),I=1,3)
      WRITE (*,102)
      READ (*,101) (V2(I),I=1,3)
C
C     TRATAMIENTO
C
      V3(1)=V1(2)*V2(3)-V2(2)*V1(3)
      V3(2)=-(V1(1)*V2(3)-V2(1)*V1(3))
      V3(3)=V1(1)*V2(2)-V2(1)*V1(2)
C
C     RESULTADO
C
      WRITE (*,103) (V3(I),I=1,3)
C
  100 FORMAT (1X,'DATOS ENTEROS DE V1 ')
  101 FORMAT (3(I3,1X))
  102 FORMAT (1X,'DATOS ENTEROS DE V2 ')
  103 FORMAT (1X,'COORD 1 ',I5,/,1X,'COORD 2 ',I5,/,
     *1X,'COORD 3 ',I5)
      STOP
      END
```

Algoritmo 19

```
DATOS ENTEROS DE V1 -23  34  10

DATOS ENTEROS DE V2  14 -90 -09

COORD 1   594
COORD 2   -67
COORD 3  1594
```

ALGORITMO 20

Dado un vector VEC compuesto por seis elementos reales de la forma ±xx.xx, contabilizar el número total de términos nulos que existen en su estructura, almacenándolo en la variable cuyo identificador es NULOS.

```
      PROGRAM PR20
C
C     PREPARACION
C
      DIMENSION VEC(6)
      WRITE (*,100)
      READ (*,101) VEC
C
C     TRATAMIENTO
C
      NULOS=0
      DO 1 I=1,6
         IF (VEC(I).EQ.0.0) THEN
            NULOS=NULOS+1
         END IF
    1 CONTINUE
C
C     RESULTADO
C
      WRITE (*,102) NULOS
C
  100 FORMAT (1X,'INTRODUCIR ELEMENTOS DEL VECTOR',/)
  101 FORMAT (6(F6.0,1X))
  102 FORMAT (1X,'TOTAL NULOS= ',I2)
      STOP
      END
```

Algoritmo 20

```
INTRODUCIR ELEMENTOS DEL VECTOR
-23.98  76.90  00.00  65.01 -67.90  12.34

TOTAL NULOS=  1

INTRODUCIR ELEMENTOS DEL VECTOR
 00.00  00.00  00.01  00.78  00.00 -34.00

TOTAL NULOS=  3
```

Bibliografía

Astor Vignau, J. *LENGUAJE DE PROGRAMACION FORTRAN IV.* PPU SA, 1983.

Day A.C. *Compatible FORTRAN.* Cambridge University Press, 1978.

Delannoy. *Apprendre à programmer en FORTRAN.* Eyrolles, 1985.

Dyck, Lawson & Smith. *FORTRAN 77, An Introduction to Structured Problem Solving.* RESTON Publishing Company, Inc.

García Merayo, F. *EL LENGUAJE FORTRAN.* Ed. Paraninfo, 2ª edición 1977.

García Merayo, F. *LECCIONES DE FORTRAN.* Facultad de Informática. Madrid, 1981.

García Merayo., F. *TEORIA DE ERRORES Y COMA FLOTANTE.* Facultad de Informática. Madrid, 1980

Kallin, Sten. *FORTRAN.* EL ATENEO Editorial. Buenos Aires, 1984.

Katzan, Harry. *FORTRAN 77.* Van Nostrand Reinhold Company, 1978.

Lignelet, P. *FORTRAN 77, Langage Fortran V.* Masson, 1984.

Marshall, G. *FORTRAN for Micros.* Newnes Technical Books.

Merchant, Michael, J. *FORTRAN 77, Language and Style.* Wadsworth Publishing Company. Belmont, California.

Metcalf, Michael. *FORTRAN Optimization.* Academic Press, 1982.

Metcalf, Michael. *EFFECTIVE FORTRAN 77.* Oxford University Press, 1985.

Miller, Alan R. *FORTRAN, Programs for Scientists and Engineers.* SYBEX. For Microsoft's FORTRAN-80 Compiler, 1982.

Obra general. American National Standard. *Programming Language FORTRAN,* ANSI X3.9-1978. ANSI Inc. 1430 Broadway, New York, New York 10018.

Radford, Arthur S. *Computer Programming in FORTRAN.* Teach yourself books. Hodder and Stoughton, 1983.

Rule, W. *FORTRAN 77, A practical Approach.* Van Nostrand Reinhold Company.

Seeds, H. L. *Structuring FORTRAN 77 for business and general applications.* Wiley and Sons.

Strohmeier, A. *FORTRAN 77, Approche Systematique Illustrée D'exemples.* Eyrolles, 1985.

Wagener, Jerrold, L. *FORTRAN 77, Principles of Programming.* Wiley & Sons, 1980.

Indice alfabético

Abrir un fichero, 117
Acceso al azar, 115
Acceso directo, 115
Acceso secuencial, 114
Alternativas, 230
Anchura del campo, 173, 185
Argumentos, 281, 282
Argumentos ficticios, 296, 302
Argumentos verdaderos, 296, 303
ASCII, 292
Asignación aritmética, 65
Asignación carácter, 256
Asignación compleja, 248
Asignación lógica, 225

Bifurcaciones, 71
Bit, 25
Blanco, 22
Blanco, COMMON, 342
Bloque IF, 227, 228, 229, 230, 233
Buffer, 359
Byte, 25

Campo, 110, 111
Carácter, 21, 145, 146, 147, 149, 152, 153, 154, 184, 193, 255
Caracteres, 21, 26
Caracteres alfabéticos, 21
Caracteres alfanuméricos, 21
Caracteres numéricos, 21
Característica, 30
Cero, 291
Ceros, 293
Cerrar un fichero, 117
Cierto, 32, 34, 216, 217, 226
Cinta, 127
Clase, 164
Clave, 115
Codificar, 17
Código de Máquina, 15
Códigos, 165
Códigos de posicionamiento, 167, 187
Códigos especiales, 168
Códigos para datos, 167
Columna, 40
Comandos del sistema, 117
Compilación, 20
Compiladores, 19
Comun, 286
Concatenación, 255
Concatenar, 254
Conector, 353
Conjunto de instrucciones, 19
Conjunto de caracteres, 21
Conjunto de datos, 55
Conjunto lineal, 56
Conjuntos carácter, 251
Conjuntos bidimensionales, 61
Conjuntos tridimensionales, 63
Conjuntos unidimensionales, 60

Constante, 22
Constante carácter, 32
Constante compleja, 243
Constante entera, 27, 33
Constante entera con exponente, 30
Constantes doble precisión, 33
Constantes FORTRAN, 22
Constantes lógicas, 32
Constantes real básica, 29, 33
Constantes real básica con exponente, 29
Constantes real normalizada, 30
Constantes real tipificada, 30
Contador, 66, 77
Control de Carro, 194
Crear, 145

DAM, 115
Dato carácter, 32
Dato complejo, 243
Dato lógico, 32
Datos, 25
Datos enteros, 165
Datos reales, 165
Decisión, 351
Delimitadores, 32
Diagramas de flujo, 74, 352
Dirección de memoria, 25
DO implícito, 140
Doble precisión, 33, 36
EBCDIC, 291

Efecto de borde, 310
Ejecutables, sentencias, 42
Elemento, 56
En línea, 279
END OF FILE, 117
Entrada/Salida, 18
Equivalente, 219
Errores lógicos, 20
Errores sintácticos, 20
Escritura, 109
Especificación de longitud, 35
Etiqueta, 41, 72
Explotación, 146
Exponente doble precisión, 33
Exponente real, 29
Expresión aritmética, 50
Expresión carácter, 255
Expresión compleja, 217, 244
Expresión de relación, 217
Expresión entera, 51
Expresión lógica, 221
Expresión mixta, 51
Expresión real, 51

Factor de escala, 198
Falso, 32, 34, 216, 217, 226
Fichero, 113
Fichero de datos, 113
Fichero externo, 358
Fichero interno, 359
Fichero secuencial, 115
Fichero vacío, 359
Fin de fichero, 136
Forma de los datos, 164
Formato, 118
Formato legible, 136
Formato libre, 118
FORTRAN Básico, 17
FORTRAN Completo, 17
Fuera de línea, 279
Función fórmula, 295
Funciones, 280, 282
Funciones externas, 299
Funciones intrínsecas, 282
Función de sentencia, 282, 295
Funciones de transferencia, 285

Gráficas arborescentes, 232

Hojas de codificación FORTRAN, 40

Identificación, 41
Instrucción de máquina, 19
Intérpretes, 19
Intersección, 219

Juego de instrucciones, 19

Lectura, 109
Lenguaje de máquina, 19
Lenguaje de programación, 15
Lenguajes de alto nivel, 16
Lenguajes ensambladores, 15
Límite inferior, 59
Límite superior, 59
Línea, 40
Línea de comentario, 40
Línea de continuación, 41
Línea de flujo, 353
Línea inicial, 40
Lista, 55, 118, 120
Lista de códigos de formato, 167
Lista de especificadores, 151
Lista de información de control, 129
Lista de opciones, 145
Lista de variables, 332
Longitud, especificación, 35

Llamada por dirección, 317
Llamada por referencia, 317
Llamada por valor, 316, 317

Mantisa, 30
Máquina de 16 bits, 26
Mega, 27
Memoria de Núcleos, 19
Memoria de Semiconductores, 19
Memoria Principal, 18
Memoria ROM, 19
Memoria Secundaria, 18
Mensajes de diagnóstico, 20
Micros de 8 bits, 26
Micros de 16 bits, 26
Módulo, 279
Multiplicación, 219

Negación, 219
No ejecutables, sentencias, 42
No equivalente, 219
Nombre, 252, 253, 302
Nombre específico, 285
Nombre genérico, 285
Nombres constantes, 260
Nombres específicos, 285
Nombres genéricos, 285
Nombres variables, 260
Número, 41
Número de iteraciones, 82
Números complejos, 217

Operador carácter, 254
Operador de caracteres, 254
Operador de concatenación, 254
Operador de relación, 216
Operadores aritméticos, 49
Operadores lógicos, 218
Ordenación de caracteres, 291
Ordenador, 17

Palabras, 25
Palabras clave, 42, 129
Por dirección, 324
Por nombre de fichero, 151
Por número de unidad, 151
Por unidad, 152, 153
Posición de memoria, 25

Primer nivel, DO, 85
Prioridad, 53
Procedimientos, 280
Proceso, 353
Programa fuente, 20
Programa objeto, 20
Programa principal, 278
Programación modular, 279
Programas de ordenador, 19
Punto fijo, 27
Punto flotante, 30

RAM, 19
Rango del DO, 80
Rebobinar, 137
Registro, 111
Registro de fin de fichero, 113, 117
Registro etiqueta, 116
Registro físico, 112
Registro lógico, 112
Registros con formato, 113
Registros sin formato, 113, 136
Retornar, 281
Retroceder la cinta, 137
ROM, 19

Salida anormal, 88
Salida normal, 88
SAM, 115
Secuencia de comparación, 292
Segundo nivel, DO, 85
Sentencia, 42
Sentencia de asignación, 65
Sentencia de entrada, 109
Sentencia de salida, 109
Sentencia terminal, 80
Sentencias auxiliares de E/S, 109, 110
Sentencias de control, 71
Sentencias de declaración, 241
Sentencias de E/S, 110
Setencias de especificación, 42
Sentencias de tipo, 36, 43, 241
Serie, 252, 259
Serie de caracteres, 252
Series, 255
Series de caracteres, 252
Sin formato, 136
Sistema de programas, 280
Slashes, 264, 342
Subíndice, 58
Subíndices, formación, 58
Subprograma, 42
Subprograma abierto, 279
Subprograma cerrado, 279
Subprogramas, 278
Subrutinas, 280
Sucesión de caracteres, 292
Sucesión de intercalación, 292

Tabla, 55
Tamaño, 164
Terminal, 352
Tipo, 283
Tipo de argumentos, 285
Tipo de función, 285
Truncado, 53
Truncamiento, 67

Unidad, 116
Unidad Aritmética y Lógica, 18
Unidad Central de Proceso, 17
Unidad de Control, 18
Unidad física, 127
Unidad lógica, 127
Unidad de programa, 278
Unidades de Entrada, 109
Unidades de Memoria, 18
Unidades de Salida, 109
Unión, 219

Valor de una expresión, 50
Variable, 23
Variable carácter, 35
Variable con subíndices, 57
Variable del DO, 80
Variable entera, 34
Variable global, 302
Variable subindicada, 57
Variables, 260
Variables complejas, 244
Variables de doble precisión, 36
Variables enteras, 327
Variables, estáticas, 332
Variables FORTRAN, 23
Variables locales, 297, 299, 331
Variables lógicas, 34
Variables reales, 34
Verdadero, 226

Zona común, 337

OTRAS OBRAS SOBRE LENGUAJES DE PROGRAMACION

PARANINFO S.A

BASIC. Curso acelerado.

BASIC. Programación de microordenadores.

CURSO COMPLETO DE BASIC (3 tomos, de venta independiente).
- Tomo I: Fundamentos
- Tomo II: Métodos
- Tomo III: Ficheros

COBOL CON PROGRAMACION ESTRUCTURADA. Curso acelerado.

44 SUPERPROGRAMAS EN BASIC.

DEPURACION DE PROGRAMAS EN BASIC.

FICHEROS EN BASIC.

LENGUAJE ADA. Sintaxis y metodología.

LENGUAJE C. Estructuras de datos. Operadores. Instrucciones. Ambiente C.

LENGUAJE COBOL CON PROGRAMACION ESTRUCTURADA.

LENGUAJE FORTH PARA MICROS.

LENGUAJES DE PROGRAMACION PARA MICROS.

LENGUAJES EN TIEMPO REAL.

MODULA 2.

PASCAL Y TURBOPASCAL.

PROBLEMAS DE PROGRAMACION.

PROGRAMACION EN FORTRAN 77.

PROGRAMACION CON EL LENGUAJE PASCAL.

PROGRAMACION EN LISP.

PROGRAMACION EN MICRO-PROLOG. Un lenguaje de la 5ª generación.

PROGRAMACION DEL Z80 CON ENSAMBLADOR.

PROGRAMAS COBOL.

RPG II. Un lenguaje de gestión.

SIMULACION CON ORDENADOR.

TECNICAS DE PROGRAMACION. Programación estructurada.

TRATAMIENTO DE TEXTOS CON BASIC.

TURBO BASIC.

TURBO C.

TURBOPASCAL. Prácticas y programas.

UNIX.